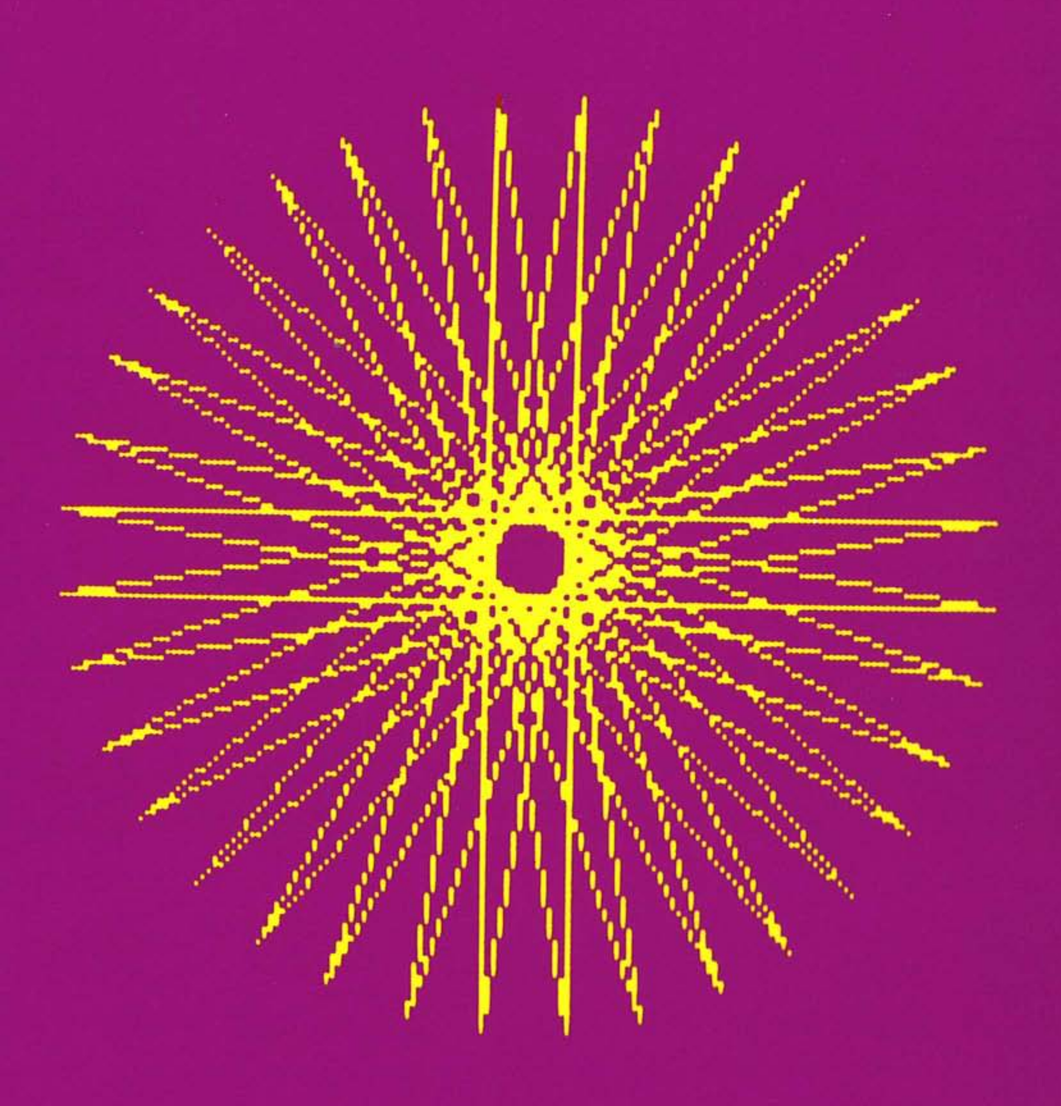
Informatik

Logo Lern- und Arbeitsbuch

westermann



Informatik

Logo Lern- und Arbeitsbuch

Jochen Ziegenbalg

westermann

1989 1988 1987 1986

Die letzte Zahl bezeichnet das Jahr der Herstellung

1. Auflage 1986
Verlagslektorat: Klaus-Peter Schrage, Rita Dittbrenner
Einbandgestalter: Heinz Neumann
Zeichnungen: Techn.-Graph. Abteilung Westermann
Titelfoto: Dieter Rixe, Braunschweig
Die Druckvorlage zu diesem Buch wurde im Laserdruck-Verfahren bei der Firma Computer Point (Tübingen) erstellt
Gesamtherstellung: poppdruck, Hannover-Langenhagen

ISBN 3-14-**112770**-0

Inhalt

Ein Hinweis zur Hardware-Abhängigkeit dieses Buches

Die Beispiele dieses Buch sind zunächst einmal für den Commodore 64 Computer geschrieben. Der größte Teil von Logo ist aber ziemlich geräteunabhängig (im Computer-Jargon sagt man: *portabel*). Ein großer Teil der Programme läuft deshalb unverändert zum Beispiel auch auf dem APPLE II Computer und dazu compatiblen Geräten. Selbst im Graphik-Bereich besteht eine gewisse Verträglichkeit zwischen dem deutschen Commodore Logo und dem deutschen Apple Logo.

Wegen dieser in der Computer-Branche ungewöhnlich hohen Portabilität von Logo kann man mit der hier verwendeten Version auch im "Mischbetrieb" mit Commodore 64 und APPLE II Computern arbeiten. Es ist insbesondere auch möglich, daß viele der Programme, die an einer Schule auf APPLE II Computern erarbeitet wurden, von Schülern zu Hause praktisch unverändert in einen Commodore 64 eingetippt und zum Laufen gebracht werden können. Auch das Umgekehrte geht natürlich.

Auf drei Abweichungen sei hier jedoch hingewiesen:

* Die "Sprite"-Befehle des Commodore 64 Logo laufen auf dem APPLE II nicht.

* Das deutsche Logo für den APPLE II verfügt zur Zeit noch nicht über das Logo-Grundwort LOKAL. Befehle wie zum Beispiel LOKAL "X oder LOKAL "ZAEHLER (oder ähnlich) sind beim Übertragen auf den APPLE II einfach wegzulassen. Allerdings werden die entsprechenden Variablen dann global (was das bedeutet, wird später erläutert).

* An Stelle des hier verwendeten LIESLISTE-Befehls ist beim APPLE II der gleichwertige Befehl EINGABE zu verwenden.

Die hier benutzte deutsche Version von Logo ist bei der Firma **IWT Verlag GmbH**, Dahlienstr. 4, D-8011 Vaterstetten / Baldham (bei München) als deutscher Sprachzusatz zum englischen Commodore Logo oder als eigenständiges deutsches Logo für den APPLE II Computer zu erhalten.

Kapitel 1: Erste Grundelememte des Programmierens

1.1 Wie man dem Computer "Logo" beibringt

Wenn man einen Taschenrechner einschaltet, dann ist er sofort betriebsbereit. Bei den meisten Heim-Computern ist das auch der Fall; allerdings kann man dann nur in der Programmiersprache BASIC, nicht aber in Logo mit ihnen arbeiten. Man sagt dann "der Computer hat BASIC im ROM". Die Abkürzung ROM steht für *read only memory*, zu deutsch "Nur-Lese-Speicher". Die Inhalte dieses Speichers können nur gelesen, nicht aber verändert werden. Auch nach dem Ausschalten des Computers gehen die Inhalte dieses Speichers nicht verloren. Man nennt ihn deshalb auch den *Festwertspeicher* des Computers. Der eigentliche Arbeitsspeicher des Computers wird als RAM bezeichnet. Dies ist eine Abkürzung für *random access memory* und bedeutet "Speicher mit wahlfreiem Zugriff". Die Inhalte dieses Speichers können beliebig abgerufen ("gelesen") und überschrieben werden.

Nur einige der allerneuesten Computer haben Logo im ROM. Bei den anderen muß man Logo von einer Diskette "laden", wenn man in dieser Sprache mit dem Computer sprechen will. Dies gilt auch für den Commodore 64. Der Ladevorgang ist nicht weiter kompliziert; er braucht nur ein klein wenig Zeit. Dies sind die einzelnen Schritte:

(1) Schalte den Computer, das Diskettenlaufwerk und den Bildschirm ein.
(2) Lege die Logo-Diskette in das Laufwerk und schliesse die Laufwerkstür.
(3) Tippe LOAD "DLOGO",8,1 und betätige die <RETURN> - Taste. Das Diskettenlaufwerk beginnt zu surren, und gleichzeitig erscheint am Bildschirm die Meldung:

    SEARCHING FOR DLOGO
    LOADING

Die Diskette läuft knapp anderthalb Minuten; dann erscheint am Bildschirm die Meldung:

    READY.

und der quadratische Blinker (englisch: cursor) ist wieder da.
(4) Tippe RUN <RETURN>-Taste. Nach wenigen Sekunden meldet sich Logo:

    COMMODORE 64 LOGO
    DEUTSCHES OVERLAY
    COPYRIGHT (C) 1984 BY IWT SOFTWARE

    WILLKOMMEN BEI LOGO!
    ?<Blinker>

Das Fragezeichen ist das sogenannte Logo-Bereitschaftszeichen (englisch: *prompt*). An diesem Zeichen können wir erkennen, daß wir im Logo-System sind und daß Logo bereitsteht, um von uns eine Eingabe entgegenzunehmen. Zum Beispiel die folgende:

    ?3*4 <RETURN>

Logo reagiert mit:

    ERGEBNIS: 12

Der oben benutzte Name DLOGO soll *deutsches Logo* bedeuten. Wenn man mit dem englischen Logo von Commodore arbeiten möchte, ist beim dritten Schritt an Stelle von DLOGO nur LOGO einzugeben. Der Ladevorgang verläuft beim englischen Commodore Logo entsprechend. Allerdings schnarrt bei dieser Version das Laufwerk wegen der Kopierschutz-Einrichtung sehr unangenehm - eine äußerst störende und irritierende Sache!

1.2 Logo im "Taschenrechner-Betrieb"

Das letzte Beispiel hat gezeigt, daß man mit Logo offenbar ähnlich wie mit einem Taschenrechner arbeiten kann. Stellen wir uns einmal vor, wie wir die Rechnung 2+3 mit einem Taschenrechner ausführen:

* Zunächst geben wir die "Rechnung" 2+3 ein.
* Es tut sich nichts. Denn da der Taschenrechner nicht "wissen" kann, ob wir 2+3 oder vielleicht 2+3.1 so gar 2+3.14 ausrechnen wollen, müssen wir ihm sagen, wann die Eingabe der zweiten Zahl beendet ist.
* Dies tun wir, indem wir das Gleichheitszeichen "=" eingeben.
* Jetzt führt der Taschenrechner die Rechnung aus und stellt das Ergebnis 5 auf seinem kleinen Anzeigefeld dar.

In Logo geht es fast genauso. Wir müssen nur an Stelle des Gleichheitszeichens die **<RETURN>-Taste** drücken. Die <RETURN>-Taste wird immer dann benutzt, wenn die Eingabe einer Zahl, eines Rechenausdruckes oder eines Wortes abgeschlossen werden soll. An Stelle des Begriffs "<RETURN>-Taste" findet man häufig auch die Bezeichnung "CARRIAGE-RETURN-Taste" oder die Abkürzung "CR-Taste". In Zukunft wird (außer in Ausnahmefällen) nicht mehr extra erwähnt werden, daß man die <RETURN>-Taste drücken muß, um die Eingabe von Daten abzuschließen. Probieren wir es gleich einmal in Logo aus:

```
2+3
ERGEBNIS: 5
```

Wir können auch längere Ausdrücke eingeben. (Die Eingabe der letzten Zahl muß mit RETURN abgeschlossen werden).

```
1+2+3+4+5+6+7+8+9
ERGEBNIS: 45
```

Ob Logo wohl die *Punkt-vor-Strich-Rechnung* beherrscht? Ein kleiner Test gibt Aufschluß darüber:

```
2+3*4
ERGEBNIS: 14
```

Wir können auch Klammern setzen:

```
(2+3)*4
ERGEBNIS: 20
```

```
(10+1)*(10-1)
ERGEBNIS: 99
```

```
(100+1)*(100-1)
ERGEBNIS: 9999
```

```
(1000+1)*(1000-1)
ERGEBNIS: 999999
```

Diese taschenrechner-ähnliche Arbeitsweise, die wir eben kennengelernt haben, nennt man den **Direktbetrieb** von Logo.

Eigentlich wollen wir ja lernen, wie man Logo-Programme schreibt, aber kleinen Problemen der folgenden Art kann man auch schon einmal im Direktbetrieb zu Leibe rücken.

Ein Beispiel: Das *Schachspiel* wurde etwa im siebenten Jahrhundert in Indien erfunden. Es wird berichtet, daß das Spiel den Herrscher Shihram, in dessen Reich es erdacht worden war, so begeisterte, daß er dem Erfinder des Schachspiels, dem Weisen *Sissa Ibn Dahir*, die Erfüllung eines Wunsches versprach.

Der Erfinder sprach die folgende Bitte aus: er wolle nur einige Reiskörner haben; und zwar eines auf dem ersten Feld des Schachbretts, zwei auf dem zweiten, vier auf dem dritten, acht auf dem vierten, sechzehn auf dem fünften, usw.; auf jedem Feld das Doppelte des vorherigen Feldes.

Der Herrscher, der sich auf eine Bitte um Gold oder Juwelen eingestellt hatte, war über so viel Bescheidenheit erstaunt. Erst nach einigem Nachdenken stellte sich heraus, daß der Wunsch weit teurer war als alle Reisvorräte, alles Gold und alle Edelsteine des Herrschers zusammengenommen.

Wir wollen einmal der Frage nachgehen, wie viele Reiskörner auf den einzelnen Feldern des Schachbretts liegen.

Feld-Nummer	Anzahl der Reiskörner
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128
9	256
10	512
11	1024
...	...
20	?
...	...
40	?
...	...
64	?

Tabelle 1.1

Aufgabe 1.1: Fülle die Tabelle aus (zum Beispiel mit Hilfe eines Taschenrechners).

Man kann die Schachbrettzahlen auf die folgende Art etwas systematischer darstellen: An Stelle von 4 können wir auch 2*2, an Stelle von 8 auch 2*2*2, an Stelle von 16 auch 2*2*2*2 schreiben, usw. Fertigen wir dazu eine neue Tabelle an:

Feld-Nummer	!	Anzahl der Reiskörner
1	!	1
2	!	2
3	!	2*2
4	!	2*2*2
5	!	2*2*2*2
6	!	2*2*2*2*2
7	!	2*2*2*2*2*2
8	!	2*2*2*2*2*2*2
9	!	2*2*2*2*2*2*2*2
10	!	2*2*2*2*2*2*2*2*2
11	!	2*2*2*2*2*2*2*2*2*2
...	!	...

Tabelle 1.2

Wenn wir wissen wollen, wie viele Reiskörner sich auf dem Feld Nr. 5 befinden, müssen wir also ein Produkt aus vier Faktoren 2 bilden. Beim Feld Nr. 11 ist ein Produkt aus 10 Faktoren, beim Feld Nr. 20 aus 19 Faktoren, beim Feld Nr. 40 aus 39 Faktoren und schließlich beim Feld Nr. 64 aus 63 Faktoren 2 zu bilden.

Wir rechnen zur Probe die Anzahl der Reiskörner auf Feld 10 nach dieser Methode im Direktbetrieb aus:

2*2*2*2*2*2*2*2*2
ERGEBNIS: 512

Die Anzahl der Körner auf Feld Nr. 20 beträgt also:

2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2
ERGEBNIS: 524288

Auf Feld Nr. 30 beträgt die Anzahl der Körner:

2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2
ERGEBNIS: 536870912

Bei Feld Nr. 31 führt diese Methode zur Körnerzahl 1073741824.

Geben wir nun für Feld Nr. 32 den Ausdruck
2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 (31 Faktoren)
in den Computer ein, so erhalten wir das Ergebnis 2.14748E9. Dabei bedeutet 2.14748E9 dasselbe wie 2.14748 mal 1 000 000 000 beziehungsweise 2 147 480 000.

Aufgabe 1.2: Berechne zur Probe die Anzahl der Körner auf Feld Nr. 32 von Hand und vergleiche die Ergebnisse.

1.3 Computer-Zahlen

Beim Übergang von Feld Nr. 31 auf Feld Nr. 32 ist folgendes passiert: Bis einschließlich zum Feld Nr. 31 wurden alle Körnerzahlen als ganze Zahlen berechnet. Dies war von der Sache her auch angemessen. Bei der Berechnung der Körnerzahl für Feld Nr. 32 wurde die größte ganze Zahl, die in Logo darstellbar ist, überschritten. Logo schaltete deshalb automatisch in die "Gleitkommadarstellung" (eine Art Dezimalzahldarstellung) um. In dieser Darstellungsform können größere Zahlen verarbeitet werden. Allerdings rechnet Logo im Gleitkommaverfahren immer nur mit den 6 wichtigsten Ziffern; die anderen Ziffern werden abgeschnitten. Deshalb sind Ergebnisse in der Gleitkommadarstellung meist nur näherungsweise richtig. (Mit den sechs "wichtigsten" Ziffern sind im allgemeinen die am weitesten links stehenden Ziffern gemeint. Zum Beispiel sind in der Zahl 390857619 die Ziffern 3 9 0 8 5 7 die sechs wichtigsten. Führende Nullen werden aber nicht zu den wichtigsten Ziffern gezählt. Die sechs wichtigsten Ziffern der Zahl 0.0390857691 sind also ebenfalls 3 9 0 8 5 7).

Merke:

(1) Logo kennt zwei Sorten von Zahlen: **ganze Zahlen** und **Gleitkommazahlen** (eine Art Dezimalzahlen).

(2) Die in Logo verfügbaren ganzen Zahlen reichen von -2147483647 bis +2147483647.

(3) Der Bereich der positiven Gleitkommazahlen reicht in Logo von 10^{-38}, also von 0.00000000000000000000000000000000000001 (37 Nullen hinter dem Dezimalpunkt) bis 10^{38}, also bis 100000000000000000000000000000000000000 (38 Nullen). Logo rechnet dabei aber nur mit den 6 wichtigsten Ziffern. Zu jeder positiven Gleitkommazahl gibt es noch die entsprechende negative Gleitkommazahl (Vorzeichenwechsel).

(4) Bei Gleitkommazahlen bedeutet der Zusatz E5 soviel wie "mal 10^5" (E steht für *Exponent*); der Zusatz N3 bedeutet "durch 10^3" (N steht für *negativer Exponent).* Beispiele:

3.586E7 steht für 35860000
2.37N5 steht für 0.0000237

(5) Im Bereiche des Zahlenrechnens weisen (fast) alle Computer und (fast) alle Programmiersprachen leider unglaublich große Schwächen auf. Dies gilt besonders für die Home- und Personal-Computer. Hier ein Beispiel im Direktbetrieb:

```
1 / 3
ERGEBNIS: 0.333333
1 / 3 - 0.333333
ERGEBNIS: 4.17232N7        (an Stelle von Null)
```

Wenn wir nun schließlich noch die Reiskörner auf Feld 40 bzw. 64 nach der oben beschriebenen Methode berechnen, so erhalten wir

auf Feld 40 5.49756E11 und
auf Feld 64 9.22338E18

Körner.

Aufgabe 1.3: Stelle die Zahl der Reiskörner auf Feld 40 und Feld 64 in der üblichen Schreibweise (ohne das Symbol E) auf.

Aufgabe 1.4: Ermittle einen (groben) Schätzwert für die Anzahl der Reiskörner, die man benötigt, um einen Rauminhalt von einem Kubikzentimeter, einem Kubikdezimeter und einem Kubikmeter zu füllen.

Aufgabe 1.5: Welche Höhe würde die Reismenge einnehmen,
(a) wenn man sie sich als "Säule" auf dem Schachbrett vorstellt?
(b) wenn man sie gleichmäßig auf der gesamten Landoberfläche der Erde verteilte? (Die Landoberfläche der Erde beträgt etwa 148 950 000 Quadratkilometer).

Aufgabe 1.6: Bestimme das durchschnittliche Gewicht eines Reiskorns (indem du zum Beispiel 100 oder 500 Reiskörner wiegst). Ermittle das Gewicht aller Reiskörner auf dem Schachbrett.

1.4 Super-Befehle / Prozeduren

Im Direktbetrieb haben wir Logo-Befehle hintereinander eingegeben. Logo verarbeitet jeden einzelnen Befehl und meldet sich dann wieder mit dem Bereitschaftszeichen. Hierzu noch ein Beispiel aus dem Graphik-Bereich. Durch die folgenden Befehle wird ein kleines Dreieck, der **"Igel"**, auf dem Bildschirm herumkommandiert. Im englischen Logo heißt der Igel *"turtle"*, also Schildkröte. Auf seinem Weg zieht der Igel eine Spur hinter sich her. Gib zunächst den Grundbefehl BILD ein. Dadurch wird der Bildschirm in den Graphik-Betriebsmodus versetzt.

```
VORWAERTS 60
RECHTS 120
VORWAERTS 60
RECHTS 120
VORWAERTS 60
RECHTS 120
```

Am Bildschirm sollte jetzt die folgende Figur zu sehen sein:

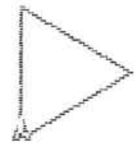


Bild 1.1: Dreieck

Aufgabe 1.7: Die Winkel beim Dreieck betragen doch 60 Grad. Warum muß sich der Igel in dem obigen Beispiel dann um 120 Grad drehen? Verfolge die einzelnen Schritte des Igels ganz genau!

Es ist allerdings etwas umständlich, wenn wir zum Zeichnen eines Dreicks jedesmal diese sechs Befehle eingeben müssen. Vielleicht kennt Logo aber auch einen Befehl DREIECK, mit dem wir das Dreieck "auf einen Schlag" zeichnen können? Probieren wir es doch einfach aus:

DREIECK

Anstatt das gewünschte Dreieck zu zeichnen, bringt Logo jedoch nur eine Fehlermeldung:

ES GIBT KEINE PROZEDUR ---- DREIECK

Logo kennt also offenbar den Befehl DREIECK nicht, aber wir brauchen deswegen die Flinte nicht ins Korn zu werfen. Die wichtigste Eigenschaft jeder "vernünftigen" Programmiersprache ist, daß man ganze Befehlsketten, wie etwa die obige, zu einem neuen Superbefehl zusammenfassen kann. Da Logo den Befehl DREIECK nicht kennt, muß es ihn also lernen. Gib folgendes ein:

LERNE DREIECK

Logo schaltet nun in den sogenannten **Editiermodus** (kurz: Editor) um und stellt oben am Bildschirm die Zeile

PR DREIECK

dar. Der Editor ist eine Art Logo-Konzeptpapier, das es uns erlaubt, Befehlsfolgen auf Vorrat aufzuschreiben; etwa so:

```
PR DREIECK
VORWAERTS  60
RECHTS  120
VORWAERTS  60
RECHTS  120
VORWAERTS  60
RECHTS  120
ENDE
```

Man schreibt diesen Text einfach wie mit einer Schreibmaschine; nur eben nicht auf Papier, sondern am Bildschirm. Wenn man sich vertippt hat, ist das nicht weiter schlimm. Man verschiebt dann einfach den Blinker mit Hilfe der Blinkerbewegungstasten bis unmittelbar rechts neben das falsche Zeichen. Die Blinkerbewegungstasten sind die mit CRSR versehenen Doppelpfeiltasten rechts unten (CRSR steht für *cursor*, das englische Wort für Blinker). Man löscht das Zeichen links vom Blinker, indem man die DEL-Taste tippt (DEL steht für *delete*, das englische Wort für löschen). Danach tippt man das richtige Zeichen ein. Natürlich geht das entsprechend auch mit ganzen Wörtern an Stelle von einzelnen Zeichen. Der Logo-Editor bietet noch viel mehr Möglichkeiten. Einige davon werden wir in Abschnitt 1.5 kennenlernen. Eine vollständige Übersicht über alle Editier-Befehle findest du im Logo-Handbuch. Fürs erste reicht die eben beschriebene Methode völlig aus.

Der Ausstieg aus dem Editor erfolgt mit Hilfe der RUN/STOP-Taste oder durch den Befehl Control-C ("Control"-Taste CTRL und Taste C gleichzeitig betätigen). Logo bringt die Meldung DREIECK GELERNT, und wir sind wieder im Direktbetrieb.

Probieren wir es aus: Auf den Befehl DREIECK zeichnet der Igel jetzt die gewünschte Figur.

DREIECK stellt einen neuen Super-Befehl dar, den wir uns selbst geschaffen haben. Solche Befehle nennt man auch **Prozeduren.** Die Buchstaben "PR" vor DREIECK stehen für Prozedur (englisch: *procedure*). Im Direktbetrieb arbeitet Logo die eingegebenen Befehle der Reihe nach ab. Wenn der Befehl ein **Grundbefehl** (englisch: *primitive*) ist, führt Logo ihn einfach aus; wenn es ein Prozedurname ist,

legt Logo eine kleine "Ehrenrunde" ein, in der es die einzelnen Schritte der Prozedur abarbeitet. Wir können uns das etwa so vorstellen:

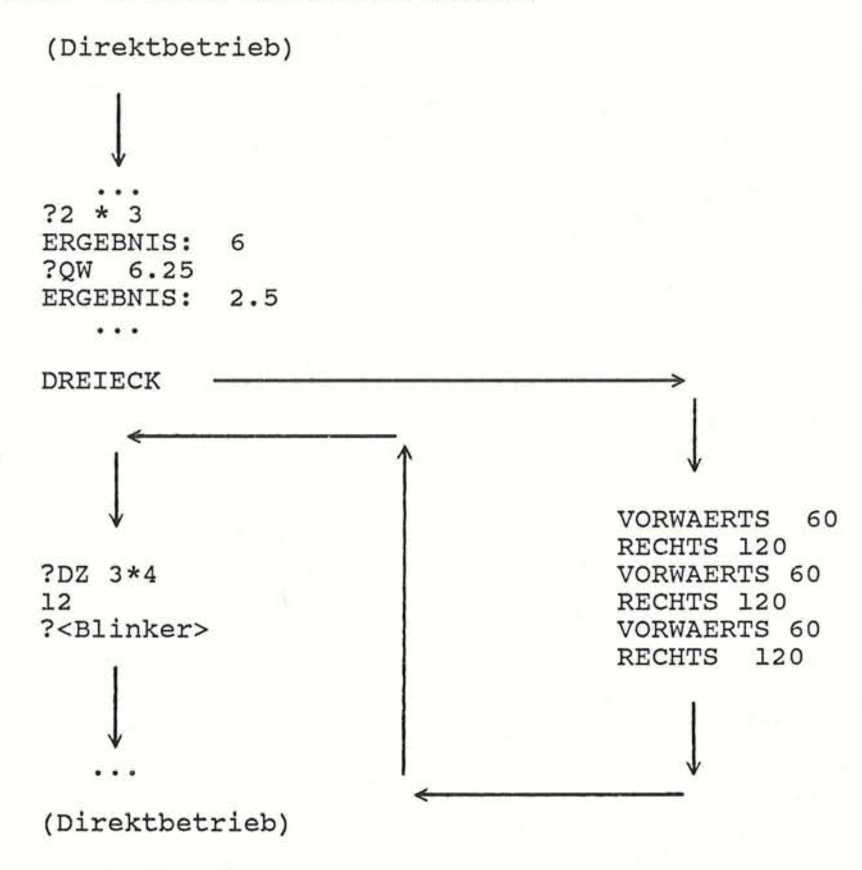


Bild 1.2: Prozeduraufruf im Direktbetrieb

Aufgabe 1.8: Schreibe nach dem obigen Muster Prozeduren zum Zeichnen von
(a) Quadraten,
(b) Fünfecken,
(c) Sechsecken.
Da bei jedem dieser Vielecke die Seiten und die Winkel gleichgroß sind, nennt man sie *regelmäßige Vielecke.*

Wir können unseren neuen Super-Befehl selbst wieder in neue "Super-Super-Befehle" einbauen. Die folgende Figur enthält drei Dreiecke als Bestandteile. Da sie wie ein Propeller aussieht, geben wir der Prozedur, mit der wir sie zeichnen, diesen Namen:

```
PR PROPELLER
DREIECK
RECHTS 120
DREIECK
RECHTS 120
DREIECK
RECHTS 120
ENDE
```

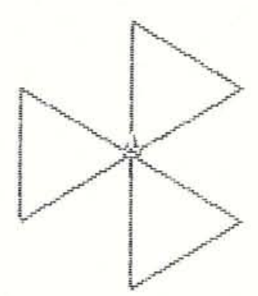


Bild 1.3: Propeller

1.5 Wie man Prozeduren abändert (Hinweise zum Logo-Editor)

Nach einer Weile möchten wir Dreiecke, Vielecke, Propeller und sonstige Figuren in verschiedenen Größen zeichnen. Auch dies ist möglich. Wir müssen den Prozeduren dann einen Eingabewert (*"Eingabeparameter"*) beifügen. Wenn wir bestimmte Prozeduren, wie etwa DREIECK, schon geschrieben haben, wäre es sehr umständlich, sie bei einer Änderung noch einmal vollständig neu einzutippen. Wir "editieren" lieber die alte Version; zum Beispiel so:

EDIT DREIECK

Logo schaltet den Editier-Bildschirm ein. Wir sehen das folgende Bild:

```
PR DREIECK
 VORWAERTS 60
 RECHTS 120
 VORWAERTS 60
 RECHTS 120
 VORWAERTS 60
 RECHTS 120
ENDE
```

Der Blinker steht auf dem "P" von PR.

Aufgabe 1.9: Ändere die Prozedur nun folgendermaßen ab:
(1) Bewege den Blinker mit der Rechtspfeil-Taste bis rechts neben das Wort DREIECK.
(2) Tippe einmal auf die Leertaste und gib danach das Wort :SEITE ein. Der Doppelpunkt unmittelbar vor SEITE ist wichtig. (Die Bedeutung des Doppelpunkts wird später noch genauer erläutert).
(3) Bewege den Blinker mit den Blinkerbewegungstasten bis unmittelbar rechts neben die Zahl 60 in der zweiten Zeile.

(4) Tippe zweimal die Löschtaste DEL.
(5) Gib die Zeichen :SEITE ein.
(6) Bewege den Blinker bis unmittelbar rechts neben die Zahl 60 in der vierten Zeile.
(7) Ersetze die Zahl 60 durch die Zeichenfolge :SEITE wie in Schritt 4 und 5.
(8) Ersetze schließlich die Zahl 60 in der sechsten Zeile durch die Zeichenfolge :SEITE wie in Schritt 6 und 7. Der Bildschirm sollte jetzt so aussehen:

```
PR DREIECK :SEITE
 VORWAERTS :SEITE
 RECHTS 120
 VORWAERTS :SEITE
 RECHTS 120
 VORWAERTS :SEITE<Blinker>
 RECHTS 120
ENDE
```

(9) Betätige jetzt die RUN/STOP-Taste.

Logo verläßt den Editor und meldet sich mit den Worten DREIECK GELERNT. Durch diese zweite Fassung der Prozedur DREIECK wurde die erste Version überschrieben. Die erste Fassung existiert nun nicht mehr. Wenn man dies hätte verhindern wollen, hätte man der neuen Version beim Editieren auch einen neuen Namen geben müssen; zum Beispiel DREIECK.VERSION.2 oder DREIECK.V2 oder DREIECK.NEU usw. Diese neuen Namen zeigen, daß man auch den Punkt in Prozedurnamen verwenden darf. Das dient der besseren Lesbarkeit. Auch andere Sonderzeichen, wie zum Beispiel das Fragezeichen, werden wir in Prozedurnamen einbauen. Man sollte aber bei der Verwendung von Sonderzeichen in Prozedurnamen vorsichtig sein und keinerlei Klammern oder Anführungszeichen in Prozedurnamen einfügen. Das **Leerzeichen** (englich: *blank* oder *space*) hat in Logo eine besondere Bedeutung: Jeder Name, jedes Wort, jede Zeichenkette wird durch das Leerzeichen beendet.

Probieren wir jetzt unsere neue Dreiecks-Prozedur aus. Zunächst geben wir BILD ein, dann:

DREIECK

Logo meldet sich mit: DREIECK WILL MEHR DATEN.

Sehr sinnvoll! Denn wir wollten ja die Seitenlänge veränderlich ("variabel") machen. Also müssen wir jetzt auch einen Wert für die Seitenlänge eingeben:

DREIECK 80

Jetzt zeichnet Logo die Figur von Bild 1.1, nur etwas vergrößert. Experimentieren wir weiter:

DREIECK 100
DREIECK 120
DREIECK 60
DREIECK 40
DREIECK 20

Man kann sich dabei das Leben erleichtern, indem man nach DREIECK 80 nur Control-P eingibt (Tastenkombination CTRL-P), den alten Eingabewert mit der

DEL-Taste löscht, den neuen Eingabewert eingibt und die <RETURN>-Taste drückt. Das Kommando Control-P wiederholt stets die letzte Eingabe des Direktbetriebs.

Diese Aufrufe der Prozedur DREIECK liefern uns die folgende Figur:

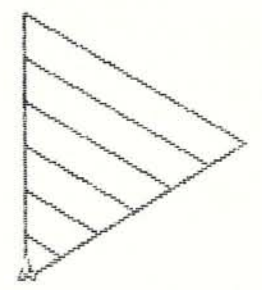


Bild 1.4: Dreiecke

Aufgabe 1.10: Du hast dich vielleicht gewundert, daß in der Prozedur DREIECK dreimal RECHTS 120 geschrieben wurde. Denn das Dreieck war ja schon nach dem dritten VORWAERTS-Befehl fertig.
(a) Entferne probehalber den Befehl RECHTS 120 in Zeile 7 und experimentiere mit der veränderten Prozedur. Gib im Direktbetrieb auch die Befehlsfolge ein, die zu Bild 1.4 geführt hat, und beobachte das jetzige Ergebnis.
(b) Korrigiere jetzt die Dreiecks-Prozedur wieder.

Merke: Wenn man geschlossene "Igel"-Figuren zeichnet, ist es meist sinnvoll, den Igel zum Schluß wieder in die Ausgangsrichtung zu bringen.

An dieser Stelle sind noch einige **Bemerkungen zum Logo-Editor** angebracht:

(1) Wie wir im obigen Beispiel gesehen haben, wurden außer der ersten und der letzten Zeile alle Zeilen um ein Zeichen nach rechts eingerückt. Dadurch werden die Prozeduren besser lesbar. Wir werden sie deshalb im folgenden immer gleich so eintippen. Wenn man die Prozedur DREIECK im Direktbetrieb nur ansehen will, ohne sie zu verändern, kann man sie sich mit dem Befehl ZEIGE DREIECK ausdrucken lassen. Auch bei diesem Ausdrucken werden alle Zeilen außer der ersten und der letzten eingerückt. Mit Hilfe des ZEIGE-Befehls kann man Prozeduren auch auf einem Drucker ausdrucken.

(2) Zur Erläuterung von Prozeduren ist es manchmal günstig, wenn man auf Zeilennummern verweisen kann. Das Commodore-Logo hat zwar keine Zeilennummern, im folgenden werden wir aber gelegentlich Zeilennummern vor die einzelnen Prozedurzeilen einfügen, falls es die Beschreibung der Prozedur erleichtert. Die Prozedur DREIECK würde dann zum Beispiel so aussehen:

```
1:   PR DREIECK :SEITE
2:    VORWAERTS :SEITE
3:    RECHTS 120
4:    VORWAERTS :SEITE
5:    RECHTS 120
6:    VORWAERTS :SEITE
7:    RECHTS 120
8:   ENDE
```

Beachte aber, daß die Zeilennummern nur der leichteren Beschreibung dienen und nicht zur Prozedur selbst gehören, also auch nicht eingetippt werden dürfen.

Wir werden auch nicht immer Zeilennummern verwenden, sondern nur dann, wenn es zur Beschreibung der Prozeduren hilfreich ist.

(3) Der besserer Lesbarkeit halber werden bei der Wiedergabe von Prozeduren in diesem Buch gelegentlich noch einige zusätzliche Leerzeichen eingefügt. Dies ist grundsätzlich unschädlich und kann auch genau so eingetippt werden. Allerdings schiebt Logo die Mehrfach-Leerstellen später eigenmächtig zu einer Leerstelle zusammen.

(4) Man kann auch mehrere Prozeduren "auf einen Schlag" editieren, indem man sie in eckige Klammern schreibt; zum Beispiel so:

EDIT [DREIECK PROPELLER FUENFECK SECHSECK]

Auch dieser lange Editier-Befehl läßt sich wieder in Kurzform als Prozedur schreiben; zum Beispiel so:

```
PR EDPG1
 EDIT  [ DREIECK PROPELLER FUENFECK SECHSECK ]
ENDE
```

EDPG1 könnte zum Beispiel "Editiere Prozedur-Gruppe 1" bedeuten. Mit dem Befehl EDPG1 können wir jetzt die vier Prozeduren in der eckigen Klammer leicht editieren.

Wir können nun auch die Prozedur PROPELLER mit einer variablen Seitenlänge versehen; etwa so:

```
PR PROPELLER  :S
 DREIECK  :S
 RECHTS  120
 DREIECK  :S
 RECHTS  120
 DREIECK  :S
 RECHTS  120
ENDE
```

Dieses Beispiel zeigt, daß beim Aufruf der Prozedur DREIECK nicht unbedingt der Variablenname SEITE verwendet werden muß. Es muß nur irgendein Eingabewert an DREIECK geliefert werden; im obigen Fall wird zum Beispiel beim Aufruf PROPELLER 50 die Variable S mit dem Wert 50 belegt, und in der der zweiten Zeile wird dann der Befehl DREIECK 50 ausgeführt.

Aufgabe 1.11: Ändere deine Prozeduren QUADRAT, FUENFECK, SECHSECK so ab, daß auch sie mit einem Eingabeparameter versehen werden.

1.6 Wie man Prozeduren verlassen kann

Die Bearbeitung der Prozedur DREIECK ist beendet, wenn alle Befehlszeilen dieser Prozedur abgearbeitet sind (vergleiche Bild 1.2). In manchen Fällen ist es jedoch notwendig, eine Prozedur zu verlassen, bevor alle ihre Befehle abgearbeitet sind. Dies ist mit Hilfe des RUECKKEHR-Befehls (kurz: RK) möglich, der nun erläutert werden soll.

Beispiel: Adam und Eva spielen ein Glücksspiel. Sie werfen mit drei Münzen: einem Pfennigstück, einem Fünf-Pfennigstück und einem Zehn-Pfennigstück. Jede der Münzen kann nach dem Wurf Wappen (W) oder Zahl (Z) zeigen. Wenn bei einem Dreierwurf die Wappen überwiegen, gewinnt Adam, sonst Eva.

Die folgende Prozedur druckt in Abhängigkeit von der Eingabe den Gewinner aus.

```
1: PR DRUCKE.SPIELAUSGANG  :X
2:  WENN  :X = "WWW  DANN DZ [ ADAM GEWINNT ] RUECKKEHR
3:  WENN  :X = "WWZ  DANN DZ [ ADAM GEWINNT ] RUECKKEHR
4:  WENN  :X = "WZW  DANN DZ [ ADAM GEWINNT ] RUECKKEHR
5:  WENN  :X = "ZWW  DANN DZ [ ADAM GEWINNT ] RUECKKEHR
6:  DZ  [ EVA GEWINNT ]
7: ENDE
```

Einige Aufrufe:

```
DRUCKE.SPIELAUSGANG "ZWW
ADAM GEWINNT

DRUCKE.SPIELAUSGANG "ZWZ
EVA GEWINNT
```

Erläuterungen zur Prozedur DRUCKE.SPIELAUSGANG: Im Gegensatz zur Prozeur DREIECK :SEITE ist die Eingabe bei DRUCKE.SPIELAUSGANG keine Zahl, sondern eine (aus drei Zeichen bestehende) Zeichenkette oder, wie man in Logo sagt, ein (dreistelliges) *Wort*. Wenn wir Logo im Direktbetrieb oder in einer Prozedur zeigen wollen, daß eine Zeichenkette als Wort (und nicht als Prozedurname) aufgefaßt werden soll, müssen wir ihr ein **Anführungszeichen** (englisch: *quote*) voranstellen. Im Gegensatz zu anderen Programmiersprachen steht das Anführungszeichen nur vor, nicht aber hinter dem Wort. Das Ende jedes Wortes ist durch das Leerzeichen gegeben. Das Anführungszeichen selbst gehört nicht zum Wort; es soll Logo nur darauf aufmerksam machen, daß jetzt ein Wort kommt. Wenn vor einer Zeichenkette ein Anführungszeichen steht, sagt man auch, die Zeichenkette sei **gequotet**. Eine kurze und treffende deutsche Ausdrucksweise ist mir dafür nicht bekannt.

Das in "WENN ... DANN ... " - Bedingungen auftretende Grundwort DANN ist an sich entbehrlich. An Stelle von Zeile 2 hätte man oben zum Beispiel auch schreiben können:

```
WENN  :X = "WWW  DZ [ ADAM GEWINNT ] RUECKKEHR
```

DZ ist eine Kurzform für den Logo-Grundbefehl DRUCKEZEILE. In Abschnitt 1.8 werden verschiedene Versionen des Druckbefehls besprochen.

Nun zum RUECKKEHR-Befehl. Wenn Logo in einer Prozedur auf diesen Befehl trifft, verläßt es die Prozedur und macht unmittelbar hinter der Stelle weiter, von

wo aus die Prozedur aufgerufen wurde. Dies kann der Direktbetrieb oder eine andere Prozedur sein. Nehmen wir einmal an, die Prozedur DRUCKE.SPIELAUSGANG wurde im Direktbetrieb mit dem Eingabewert WZW aufgerufen. Dann können wir uns dies ähnlich wie in Bild 1.2 folgendermaßen veranschaulichen.

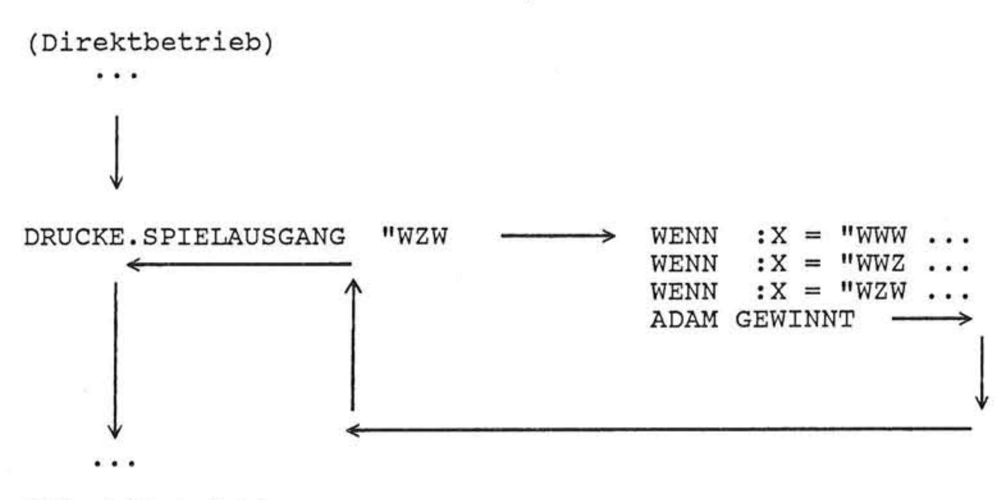


Bild 1.5: Das Verlassen von Prozeduren

Aufgabe 1.12: Entferne die RUECKKEHR-Befehle aus der Prozedur DRUCKE.-SPIELAUSGANG und rufe die Prozedur mit verschiedenen Eingabewerten auf. Beachte genau, was passiert. Korrigiere die Prozedur wieder, wenn du mit dem Experimentieren fertig bist.

1.7 Zeichketten und Wörter

In der Prozedur DRUCKE.SPIELAUSGANG haben wir erste Beispiele für Zeichenketten (Wörter) kennengelernt. Wir wollen das Arbeiten mit Wörtern hier noch etwas vertiefen. Probiere folgendes im Direktbetrieb aus:

XYZ
ES GIBT KEINE PROZEDUR ---- XYZ

"XYZ
ERGEBNIS: XYZ (dreistelliges Wort)

"XYZ"
ERGEBNIS: XYZ" (vierstelliges Wort; das hinter XYZ stehende Anführungszeichen ist Bestandteil des Wortes)

DRUCKE.SPIELAUSGANG WZZ
ES GIBT KEINE PROZEDUR ---- WZZ

Da die Zeichenkette WZZ nicht gequotet war und auch keinen Logo-Grundbefehl darstellt, versuchte Logo, sie als den Namen einer Prozedur zu interpretieren. Aber

es war auch keine Prozedur unter dem Namen WZZ definiert. Daher also die Fehlermeldung.

Merke: Logo versucht, jede Eingabe als Grundbefehl oder als Prozedurnamen zu interpretieren und entsprechend auszuführen. Gequotete Eingaben werden dagegen als Zeichenketten (Wörter) registriert und unverändert zurückgegeben.

Jedes Wort (jede Zeichenkette) wird durch das Leerzeichen beendet.

Logo verfügt über eine Reihe von Grundbefehlen, um Wörter zu verarbeiten. Hier einige Beispiele für die wichtigsten davon.

```
ERSTES "XYZ        (kurz: ER "XYZ )
ERGEBNIS: X

OHNEERSTES "XYZ    (kurz: OE "XYZ )
ERGEBNIS: YZ

LETZTES "XERXES    (kurz: LZ "XERXES )
ERGEBNIS: S

OHNELETZTES "XERXES (kurz: OL "XERXES )
ERGEBNIS: XERXE
```

Mit Hilfe des Logo-Grundbefehls WORT lassen sich zwei Zeichenketten zu einer einzigen Kette zusammenfassen.

```
WORT "SCHNABEL "TIER
ERGEBNIS: SCHNABELTIER
```

Logo kennt auch "Zahlwörter":

```
OHNELETZTES "7410
ERGEBNIS: 741
```

Man kann die Befehle auch miteinander kombinieren:

```
ERSTES OHNEERSTES "XAVER
ERGEBNIS: A

WORT ERSTES "LUENEBURG LETZTES "LUENEBURG
ERGEBNIS: LG
```

Die Auswertung des letzten Aufrufs erfolgt so:

```
ERSTES "LUENEBURG ergibt L
LETZTES "LUENEBURG ergibt G
```

Diese beiden Zeichen werden vom Grundbefehl WORT zu der Zeichenkette LG zusammengefügt.

1.8 Druckbefehle

Logo kennt verschiedene Formen des Druckbefehls. Probiere folgendes aus:

```
DRUCKE 4 * 7            (kurz: DR  4 * 7 )
28?<Blinker>
```

```
DRUCKEZEILE 4 * 7       (kurz: DZ  4 * 7 )
28
?<Blinker>
```

Nach dem DRUCKE-Befehl kommt unmittelbar das Logo-Bereitschaftszeichen (Fragezeichen) und der Blinker. Der Befehl DRUCKEZEILE (kurz DZ) bewirkt, daß nach dem Ausdruck der Daten ein **Zeilenvorschub** und **Wagenrücklauf** (englisch: *linefeed* und *carriage return*) eingeschoben wird. Im folgenden werden, außer in seltenen Ausnahmefällen, das Bereitschaftszeichen und der Blinker nicht mehr dargestellt.

Man kann auch mehrere Dinge mit einem Druck-Befehl ausdrucken. Damit Logo weiß, wann Schluß ist, muß man dann allerdings den gesamten Druckbefehl in runde Klammern einschachteln. Ein Beispiel:

```
( DRUCKEZEILE  "12  "+  "987  "=  12+987  )
12 + 987 = 999
```

Der letzte DRUCKEZEILE-Befehl hatte fünf Eingaben: das zweistellige Wort 12, das einstellige Wort +, das dreistellige Wort 987, das einstellige Wort = und den Rechenausdruck 12+987. Die Anführungszeichen gehörten nicht zu den Wörtern selbst; sie sollten Logo nur andeuten, daß es sich bei dem, was folgte, um Wörter handelte.

Beim DRUCKEZEILE-Befehl druckt Logo eine Leerstelle hinter jede einzelne Ausgabe; beim DRUCKE-Befehl läßt Logo dagegen keine Lücken. Noch ein Beispiel zum Kontrast:

```
( DRUCKE  "12  "+  "987  "=  12+987  )
12+987=999
```

Falls man, wie in der Prozedur DRUCKE.SPIELAUSGANG, mehrere Wörter ausdrucken und der besseren Lesbarkeit halber auf die Anführungszeichen verzichten will, dann kann man diese Wörter auch in eckige Klammern schreiben. Der gesamte Inhalt der eckigen Klammern (einschließlich der Klammern selbst) wird dann als eine einzige Eingabe zu dem Druckbefehl gewertet. Deshalb braucht man dann auch keine runden Klammern. Ein Beispiel:

```
DRUCKEZEILE [ EIN MOPS KAM IN DIE KUECHE ]
EIN MOPS KAM IN DIE KUECHE
```

Merke: Logo läßt bei Druckbefehlen die äußersten eckigen Klammern weg.

Vergleiche aber hiermit:

```
DRUCKEZEILE [ GESCHACHTELTE [ [ ECKIGE ] KLAMMERN ] ]
GESCHACHTELTE [ [ ECKIGE ] KLAMMERN ]
```

1.9 Funktionen

Eine bestimmte Art von Prozeduren ist in Logo besonders wichtig. Man nennt sie *Funktionen.* Zwischen Logo-Funktionen und mathematischen Funktionen (**Zuordnungen**) gibt es der Sache nach praktisch keinen Unterschied; sie unterscheiden sich nur geringfügig in der Schreibweise.

Eine Funktion, die in der Mathematik eine große Rolle spielt, ist die *Quadratwurzel*-Funktion. Man beschreibt sie meist folgendermaßen mit Hilfe des "Wurzel"-Zeichens:

$$x \longrightarrow \sqrt{x}$$

In Worten ausgedrückt, wird diese Zeile meist so gelesen: "x wird auf Wurzel x abgebildet" oder "der Zahl x wird die Wurzel von x als Funktionswert zugeordnet".

Die Quadratwurzel-Funktion gibt es auch in Logo. Sie heißt einfach QW (für Quadratwurzel). Wenn man die Quadratwurzel der Zahl 6.25 bestimmen möchte, gibt man einfach

QW 6.25

ein, und Logo antwortet mit

ERGEBNIS: 2.5

Manche Funktionen besitzen gar keinen eigenen Namen, sondern sie werden einfach als Rechenausdruck hingeschrieben; zum Beispiel die folgende "Funktion":

$$x \longrightarrow 1 / x$$

Gelegentlich möchte man aber auch solche "anonymen" Funktionen mit einem richtigen Namen versehen, um sie besser ansprechen zu können. Wir könnten der Funktion aus dem letzten Beispiel etwa den Namen EINSDURCH geben. In der Mathematik ist jedoch der Name INVERS gebräuchlicher:

$x \longrightarrow$ INVERS(x) (mit INVERS(x) = 1 / x)

Man kann sich eine Funktion wie eine Maschine vorstellen: die Funktion wird mit einer "Eingabe" gefüttert, sie verarbeitet diese Eingabe entsprechend der Funktionsvorschrift und gibt das Ergebnis zum Schluß als Funktionswert aus. Die folgenden schematischen Darstellungen sollen dies veranschaulichen:

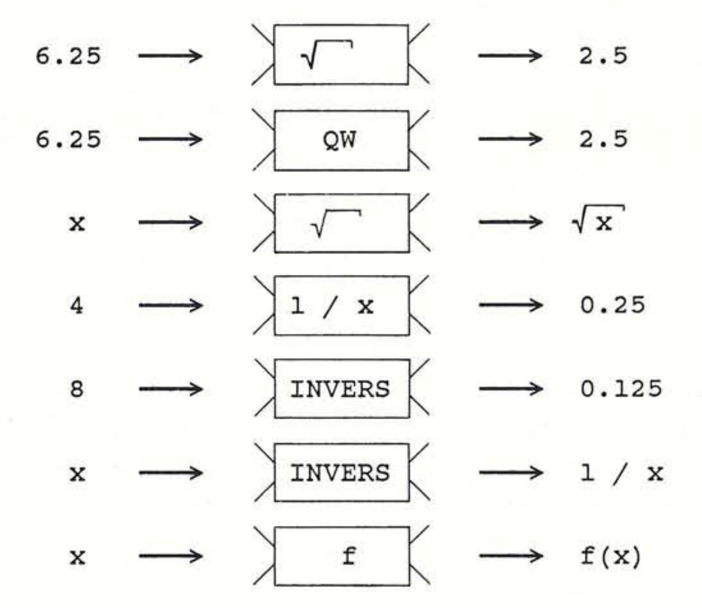


Bild 1.6: Funktionen als Maschinen

Funktionen sind deshalb so wichtig, weil man sie hervorragend verketten, d.h. aneinanderhängen, kann. Man macht dabei die Ausgabe der ersten Funktion einfach zur Eingabe der zweiten Funktion; im Schema:

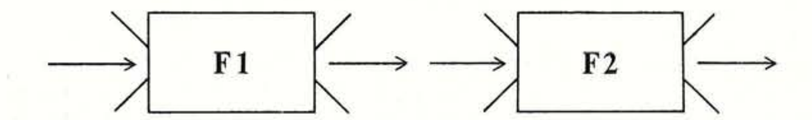


Bild 1.7: Funktionsverkettung

Für die Funktionsverkettung gibt es verschiedene Schreibweisen. In der Mathematik schreibt man die Verkettung der Funktion f mit der Funktion g meist folgendermaßen:

x ⟶ [g o f] ⟶ g(f(x))

Ist zum Beispiel f die Funktion QW und g die Funktion INVERS, so könnte ein Verkettungsbeispiel (mit Näherungwerten) etwa lauten:

2 ⟶ [INVERS o QW] ⟶ INVERS(QW(2)) =
INVERS(1.41421) = 0.707108

Im Bild:

2 ⟶ [QW] ⟶ 1.41421 ⟶ [INVERS] ⟶ 0.707108

Bild 1.8: Verkettungsbeispiel

Aufgabe 1.13: (a) Begründe, warum INVERS(QW(x)) und QW(INVERS(x)) für jede positive Zahl x übereinstimmen. Man sagt, die Funktionen INVERS und QW sind bei der Verkettung *vertauschbar*.
(b) Überprüfe anhand einiger Zahlenwerte, ob dies beim Arbeiten mit dem Taschenrechner bzw. Computer auch so ist.
(c) Gib zwei Funktionen an, die in bezug auf die Verkettung *nicht* vertauschbar sind.

Logo unterstützt das Programmieren mit Funktionen wie kaum eine andere Programmiersprache. Dazu gehört zum Beispiel, daß man bei der Ein- und Ausgabe nicht etwa nur auf Zahlenwerte beschränkt ist. Logo-Funktionen können auch Wörter und zusammengesetzte Daten verarbeiten. Die in Abschnitt 1.7 behandelten Logo-Grundbefehle ERSTES, OHNEERSTES, LETZTES, OHNELETZTES und WORT sind zum Beispiel Logo-Funktionen zur Verarbeitung von Wörtern. Hier noch als Beispiel eine Dreifach-Verkettung:

```
OHNELETZTES OHNELETZTES OHNELETZTES "TRIER
ERGEBNIS: TR
```

In schematischer Darstellung sieht der letzte Aufruf (mit der Abkürzung OL für OHNELETZTES) folgendermaßen aus:

TRIER → [OL] → TRIE → [OL] → TRI → [OL] → TR

Bild 1.9: Dreifach-Verkettung

Aufgabe 1.14: Zu welchem Ergebnis führt der folgende Aufruf:
WORT ERSTES "TRIER LETZTES "TRIER

Weiterhin gehört zu einem umfassenden Funktionskonzept natürlich auch die Möglichkeit, daß man sich seine eigenen Funktionen selbst schreiben kann. Dies soll im folgenden an einigen weiteren Beispielen erläutert werden.

Beispiel: In vielen Zusammenhängen benötigt man den Absolutbetrag einer Zahl. Wenn die Zahl positiv oder gleich Null ist, dann unterscheidet sie sich nicht von ihrem Absolutbetrag; wenn sie negativ ist, erhält man ihrem Absolutbetrag, indem man sie mit -1 multipliziert.

Die Prozedur, in der dieser Absolutbetrag ermittelt wird, soll ABS heißen. Da wir verschiedene Versionen betrachten werden, hängen wir an den Namen jeweils eine Versionsnummer V1, V2, ... an. Hier eine erste Version, an der zunächst einmal gezeigt wird, wie man es *nicht* machen soll:

```
PR ABS.V1   :X
 WENN  :X < 0  DANN DRUCKEZEILE -:X  SONST DRUCKEZEILE :X
ENDE
```

Einige Aufruf-Beispiele:

```
ABS.V1  -4.7
4.7

ABS.V1  8.5
8.5
```

Diese Version von ABS liefert zwar den jeweils gefragten Absolutbetrag, man kann sie aber nicht mit anderen Funktionen verketten. Beim Aufruf QW ABS.V1 -5 druckt Logo zunächst die Zahl 5 aus und bringt dann die Fehlermeldung ABS.V1 KEINE RUECKGABE.

Der Fehler bei der Prozedur ABS.V1 besteht darin, daß sie zwar den richtigen Wert ermittelt, ihn dann aber nur ausdruckt und nicht weiterleitet. ABS.V1 ist zwar eine Prozedur, aber keine Funktion.

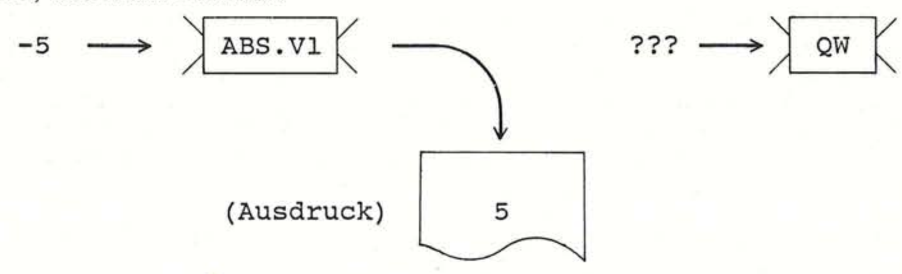


Bild 1.10: fehlende Funktionsverkettung

Die Weiterleitung von Funktionswerten erfolgt in Logo mit Hilfe des RUECKGABE-Befehls. Die Kurzform für RUECKGABE lautet RG. Hier ist eine Version von ABS, bei der der ermittelte Funktionswert richtig weitergeleitet wird:

```
PR ABS.V2  :X
 WENN  :X < 0  DANN  RUECKGABE  -:X  SONST  RUECKGABE  :X
ENDE
```

ABS.V2 3.25
ERGEBNIS: 3.25

ABS.V2 -4.75
ERGEBNIS: 4.75

QW ABS.V2 -5
ERGEBNIS: 2.23606

Die Version ABS.V2 leitet ihre Werte ordentlich (mit Hilfe des RUECKGABE-Befehls) weiter; in schematischer Darstellung:

-5 ⟶ ABS.V2 ⟶ 5 ⟶ QW ⟶ 2.23606

Bild 1.11: richtige Funktionsverkettung

Ein entscheidendes Merkmal des RUECKGABE Befehls ist, daß er, ähnlich wie der RUECKKEHR-Befehl, das Verlassen der gerade durchlaufenen Prozedur bewirkt. Deshalb kann eine zur zweiten Version gleichwertige Version auch so geschrieben werden:

```
PR ABS.V3  :X
 WENN :X < 0  DANN  RUECKGABE  -:X
 RUECKGABE  :X
ENDE
```

Im Kontrast hierzu würde die folgende "Druck"-Version zu unerwünschten Ergebnisse führen (denn der DRUCKEZEILE-Befehl hat nicht zur Folge, daß die Prozedur verlassen wird). Auch ABS.V4 ist keine Funktion.

```
PR ABS.V4  :X
 WENN :X < 0   DANN  DRUCKEZEILE  -:X
 DRUCKEZEILE  :X
ENDE
```

ABS.V4 5
5

ABS.V4 -17
17
-17

Aufgabe 1.15: Korrigiere die Version ABS.V4 durch Hinzufügen eines einzigen Grundwortes so, daß das im letzten Beispiel dokumentierte Mißgeschick vermieden wird. (Hinweis: das Grundwort SONST ist nicht gemeint).

Aufgabe 1.16: Die Version ABS.V3 paßt sehr gut mit den Grundprinzipien von Logo zusammen. Wir wollen sie in Zukunft benutzen. Ändere sie im Editor ab, so daß ihr Name nur noch ABS ist.

Aufgabe 1.17: Zeichne (von Hand) das Funktionsschaubild der Funktion ABS.

Funktionen können beliebig viele Eingaben haben. Hier eine Funktion mit zwei Eingaben:

```
PR ABSTAND  :X  :Y
 RUECKGABE  ABS  ( :Y - :X )
ENDE
```

Das Ergebnis dieser Funktion ist immer positiv.

ABSTAND 2 5
ERGEBNIS: 3

ABSTAND 5 2
ERGEBNIS: 3

Beachte folgendes Beispiel:

ABSTAND 5 -8<RETURN>
Logo antwortet: ABSTAND WILL MEHR DATEN

Logo hat zuerst den Rechenausdruck 5 -8 (= -3) ausgewertet. Damit fehlte dann der zweite Eingabewert. Um dies zu verhindern, hätte man die Eingabewerte (mit runden Klammern) einklammern müssen. Es genügt auch, den mit dem Minuszeichen versehenen Eingabewert einzuklammern:

ABSTAND 5 (-8)
ERGEBNIS: 13

Die folgende Funktion verdoppelt ihren Eingabewert:

```
PR VERDOPPLUNG  :X
 RUECKGABE  2 * :X
ENDE
```

VERDOPPLUNG 5
ERGEBNIS: 10

VERDOPPLUNG 2.15
ERGEBNIS: 4.3

Wir können die neuen Befehle auch mit den Befehlen des Logo-Grundwortschatzes verknüpfen:

2 + VERDOPPLUNG 3
ERGEBNIS: 8

QW VERDOPPLUNG 8
ERGEBNIS: 4

Was der Quadratwurzelfunktion recht ist, ist der Verdopplungs-Funktion billig:

VERDOPPLUNG VERDOPPLUNG 3
ERGEBNIS: 12

VERDOPPLUNG VERDOPPLUNG VERDOPPLUNG VERDOPPLUNG 2
ERGEBNIS: 32

Die folgende Funktion gibt das Volumen eines Würfels von einer bestimmten Kantenlänge als Funktionswert zurück.

```
PR WUERFELVOLUMEN  :KANTE
 RUECKGABE  :KANTE * :KANTE * :KANTE
ENDE
```

Ein Beispiel:

WUERFELVOLUMEN 3
ERGEBNIS: 27

Wir können die Funktionen VERDOPPLUNG und WUERFELVOLUMEN beliebig miteinander kombinieren:

WUERFELVOLUMEN VERDOPPLUNG 5
ERGEBNIS: 1000

VERDOPPLUNG WUERFELVOLUMEN 5
ERGEBNIS: 250

Diese Ergebnisse kommen folgendermaßen zustande: Im ersten Beispiel ergibt VERDOPPLUNG 5 den Wert 10, und dieser Wert wird als Eingabewert in die Funktion WUERFELVOLUMEN eingespeist; WUERFELVOLUMEN 10 ergibt dann 1000.

Dagegen ergibt WUERFELVOLUMEN 5 im zweiten Beispiel zunächst 125. Dieser Wert wird als Eingabewert in die Funktion VERDOPPLUNG eingespeist. Als Ergebnis erhalten wir 250.

Bildlich können wir uns diese Auswertungsvorgänge folgendermaßen vorstellen:

Erstes Beispiel:

5 ⟶ VERDOPPLUNG ⟶ 10 ⟶ WUERFELVOLUMEN ⟶ 1000

Zweites Beispiel:

5 ⟶ WUERFELVOLUMEN ⟶ 125 ⟶ VERDOPPLUNG ⟶ 250

Bild 1.12: nicht vertauschbare Funktionen

Aufgabe 1.18: (a) Verkette je zwei der Funktionen QW, ABS, VERDOPPLUNG und WUERFELVOLUMEN miteinander (experimentiere dabei mit verschiedenen Zahlenwerten). Stelle fest, welche der Funktionen nicht vertauschbar sind.
(b) Kann man mit Hilfe des Computers durch Probieren feststellen, welche der Funktionen vertauschbar sind?
(c) Begründe durch algebraische Umformungen, welche der obigen Funktionen vertauschbar sind.

Aufgabe 1.19: Schreibe Funktionen zur Verdreifachung, Vervierfachung, Verfünffachung der Eingabewerte.

Aufgabe 1.20: (a) Schreibe eine Funktion PLUS1 :X, die folgendes bewirkt:

```
PLUS1 3
ERGEBNIS: 4
PLUS1 17
ERGEBNIS: 18
```

(b) Schreibe entsprechende Funktionen PLUS2 und PLUS3.

Aufgabe 1.21: (a) Schreibe eine Funktion SIGNUM :X mit der folgenden Eigenschaft:

SIGNUM :X = 1 , falls :X > 0 ist;
SIGNUM :X = 0 , falls :X = 0 ist;
SIGNUM :X = -1 , falls :X < 0 ist.
(SIGNUM heißt soviel wie "Vorzeichen").

(b) Zeichne das Funktionsschaubild der Funktion SIGNUM.

Aufgabe 1.22: Schreibe eine Funktion
ABSTAND.IN.DER.EBENE :X1 :Y1 :X2 :Y2 ,
die als Ausgabewert den Abstand des Punkts (:X1 / :Y1) vom Punkt (:X2 / :Y2) im üblichen Koordinatensystem hat.

Aufgabe 1.23: Schreibe eine Funktion LINEAR :M :B :X zur Berechnung der Werte des Ausdrucks m*x + b.

Aufgabe 1.24: Schreibe eine Funktion QUADRATISCH :A :B :C :X zur Auswertung des Ausdrucks a*x*x + b*x + c.

Aufgabe 1.25: Schreibe Funktionen ZWEITES, DRITTES, VORLETZTES, VORVORLETZTES, mit denen du die entsprechenden Zeichen aus einem Wort herauspicken kannst; zum Beispiel:

```
ZWEITES "HUGO
ERGEBNIS: U
VORVORLETZTES "DONAU
ERGEBNIS: N
```

Merke: (1) *Prozeduren* können verschiedene Dinge tun: Speicherplätze belegen oder löschen, Daten ausdrucken, Daten von der Diskette lesen oder auf eine Diskette schreiben, Figuren auf den Bildschirm zeichnen, den Lautsprecher des Computers zum Tönen bringen und vieles mehr. Alles dies bezeichnet man als *Nebenwirkungen*, die den Zustand des Computers und seiner "Peripherie"-Geräte verändern.

(2) *Funktionen* sind spezielle Prozeduren (so wie Quadrate spezielle Rechtecke sind). Die Hauptaufgabe einer Funktion besteht darin, ihren Funktionswert zu ermitteln und weiterzuleiten. Jede Logo-Funktion ist auch eine Logo-Prozedur, die Umkehrung ist aber nicht richtig. Die Logo-Funktion WUERFELVOLUMEN kann auch als Logo-Prozedur bezeichnet werden, die Logo-Prozedur DREIECK ist aber keine Logo-Funktion. Im Logo-Editor beginnen sowohl Logo-Funktionen als auch Logo-Prozeduren mit den Buchstaben PR (für Prozedur).

(3) Der RUECKGABE-Befehl bewirkt die Weiterleitung des ermittelten Funktionswertes; und zwar an die Stelle, von der aus die Funktion aufgerufen wurde. Dies kann der Direktbetrieb oder eine andere Funktion (oder Prozedur) sein. Mit der Rückgabe des Funktionswertes wird die Funktion verlassen.

(4) Logo-Funktionen sind beliebig verkettbar. Sie entsprechen deshalb ziemlich genau den Funktionen in der Mathematik.

(5) Wird eine Funktion im Direktbetrieb aufgerufen, so druckt Logo zuerst das Wort "ERGEBNIS:" und dann den Funktionswert.

(6) Druckbefehle "fressen" den Funktionswert auf; er steht dann nicht mehr zur Weiterverarbeitung zur Verfügung (siehe ABS.V1 in Bild 1.10).

(7) Funktionen spiegeln sehr genau das "EVA"-Grundprinzip der Datenverarbeitung wieder:

Eingabe ⟶ Verarbeitung ⟶ Ausgabe

Funktionen können mehrere Eingabewerte, aber stets nur einen Ausgabewert haben. Allerdings kann die Ausgabe auch ein ganzer "Sack" voller Daten sein, wie wir später im Zusammenhang mit den "Listen" sehen werden.

Dem Prinzip von "Eingabe, Verarbeitung und Ausgabe" muß jede Programmiersprache in irgendeiner Weise Rechnung tragen. Bei Programmiersprachen, in denen man keine eigenen Prozeduren schreiben kann (BASIC ist zum Beispiel eine solche Sprache), wird die Eingabe üblicherweise mit Hilfe des INPUT-Befehls und die Ausgabe mit Hilfe des PRINT-Befehls vorgenommen. Solche Befehle gibt es in Logo natürlich auch. Das Gegenstück zu PRINT, die Druckbefehle von Logo, haben wir in Abschnitt 1.8 bereits kennengelernt. Dem INPUT-Befehl von BASIC entspricht in Logo der LIESLISTE-Befehl (bzw. beim APPLE Computer der EINGABE-Befehl). In Kapitel 5 über Wertetafeln ist in der Prozedur WERTETAFEL.DIALOG ein typisches Beispiel für die Verwendung des LIESLISTE-Befehls gegeben.

In den meisten Fällen gibt es aber in Logo weit bessere Möglichkeiten, Eingabewerte bereitzustellen; nämlich über die Eingabeparameter von Prozeduren. Wir wollen dies am Beispiel der Funktion SIGNUM (siehe Aufgabe 1.21) verdeutlichen.

Nehmen wir an, wir wollen den Wert von SIGNUM an der Stelle 6.25 bestimmen. Die natürlichste Sache von der Welt ist, SIGNUM 6.25 einzutippen und sich das Er-

gebnis ausgeben zu lassen. Entsprechend gehen wir ja zum Beispiel auch vor, wenn wir den Wert der Quadratwurzel-Funktion an der Stelle 6.25 ermitteln wollen. Warum sollte es bei selbstdefinierten Funktionen anders sein? Es gibt keinen Grund dafür.

In BASIC gibt es keine Möglichkeit, in selbstdefinierten Funktionen Fallunterscheidungen einzubauen. Man könnte SIGNUM also nicht als BASIC-Funktion formulieren, sondern man müßte ein Programm schreiben, das mit Hilfe des Befehls RUN aufzurufen wäre. Der Dialog müßte in BASIC etwa folgendermaßen verlaufen:

```
Benutzer:  RUN
Programm:  WERTER BENUTZER; HIER MELDET SICH DAS PROGRAMM
           SIGNUM.
           WELCHE ZAHL SOLL ICH DENN VERARBEITEN:
Benutzer:  6.25
Programm:  ICH HABE NUN SIGNUM VON 6.25 BERECHNET.
           HIER IST DER WERT:  +1
```

Das ganze ist natürlich etwas dick aufgetragen, aber in ähnlicher Weise müßte es schon ablaufen. Vielleicht scheint dir das sogar im ersten Moment Spaß zu machen. Aber überlege einmal, ob du diesen ganzen "Zauber" jedesmal mitmachen möchtest, wenn du zum Beispiel die Quadratwurzel oder den Sinus einer Zahl ausrechnen willst. Der Informationsgehalt des obigen Beispiels liegt in einem einzigen Ausgabewert. Angesichts dieser Tatsache ist der obige "Dialog" doch ziemlich aufwendig.

Was aber das Schlimmste ist: das Verketten von Funktionen wird durch diesen geschwätzigen Dialog praktisch unmöglich gemacht.

1.10 Was ist ein Programm?

In Logo lautet die Antwort kurz und bündig: Jede Prozedur (und somit auch jede Funktion) ist ein Programm. In den meisten Programmiersprachen hat man zu einem bestimmten Zeitpunkt nur ein einziges Programm im Speicher, und um es aufzurufen, muß man "RUN" (oder etwas ähnliches) eintippen. In Logo kann man zu jedem Zeitpunkt viele Prozeduren gleichzeitig im Speicher haben, und jede Prozedur wird dadurch aufgerufen, daß man ihren Namen und die jeweiligen Eingabewerte eingibt.

1.11 Der Arbeitsspeicher

Mit dem Befehl ZEIGE ALLES kann man sich zu jedem Zeitpunkt einen Überblick darüber verschaffen, was alles im Speicher ist. Mit Hilfe des Befehls ZEIGE TITEL (kurz: ZT) kann man alle bekannten Prozedurnamen auflisten. Wenn wir Logo mit Beginn dieses Kapitels geladen und seitdem nicht mehr ausgeschaltet haben, liefert ZEIGE TITEL in etwa den folgenden Ausdruck:

```
PR DREIECK  :SEITE
PR PROPELLER  :S
PR DRUCKE.SPIELAUSGANG  :X
PR ABS.V1  :X
PR ABS.V2  :X
PR ABS.V3  :X
PR ABS.V4  :X
PR ABSTAND  :A  :B
PR VERDOPPLUNG  :X
PR WUERFELVOLUMEN  :KANTE
```

Dazu kommen eventuell noch weitere selbstgeschriebene Prozeduren.

1.12 Wie man seine Arbeit dauerhaft auf einer Diskette abspeichert

Würden wir jetzt den Computer ausschalten, wären alle diese Prozeduren verloren, und wir müßten sie neu eintippen, wenn wir sie wieder benötigen.

In jeder Programmiersprache gibt es Befehle, um Programme auf einem (relativ) dauerhaften Speichermedium abzuspeichern. Beim Commodore 64 ist das die Diskette. Mit dem Befehl

BEWAHRE "BEISPIELE

speichern wir den gesamten Arbeitsspeicher im augenblicklichen Zustand unter dem Namen BEISPIELE auf der im Laufwerk befindlichen Diskette ab.

An Stelle des Namens BEISPIELE könnte auch jeder andere Name verwendet werden. Vorsicht mit Namen, unter denen sich schon eine Datei auf der Diskette befindet! Diese Datei würde bei Verwendung desselben Namens überschrieben und dadurch gelöscht werden.

Mit dem Befehl IH (kurz für INHALT) können wir uns anschauen, was schon alles auf der Diskette abgespeichert worden ist. Wir finden die Datei, in der wir gerade unsere Prozeduren abgespeichert haben, unter dem Namen BEISPIELE.LOGO wieder. Den Zusatz ".LOGO" hat Logo selbständig angehängt. Er dient nur zur Identifizierung der verschiedenen Dateien und darf beim Laden der Datei nicht eingegeben werden.

Das Laden unserer Prozeduren (zum Beispiel nach dem Ausschalten des Computers und einem neuen Start mit Logo) erfolgt mit dem Befehl

LADE "BEISPIELE

Manche der obigen Prozeduren dienten nur der Demonstration, wie man es nicht machen soll. Diese Prozeduren willst du vielleicht nicht speichern. Du kannst sie mit Hilfe des VERGISS-Befehls löschen; zum Beispiel:

VERGISS ABS.V1
VERGISS ABS.V4

Aufgabe 1.26: Lösche alle weiteren überflüssigen Prozeduren und behalte genau diejenigen im Speicher, welche auf die Diskette sollen. Speichere die verbleibenden Prozeduren unter einem sinnvollen Namen ab.

Kapitel 2: Grundlegende Programmiertechniken

Wir kehren noch einmal zum Schachproblem aus Kapitel 1 zurück. Der Berechnung der Schachbrettzahlen liegt der Vorgang des Verdoppelns zugrunde, den wir im vorigen Kapitel durch die Funktion VERDOPPLUNG in eine feste Form gegossen haben:

```
PR VERDOPPLUNG  :X
 RUECKGABE  2 * :X
ENDE
```

Aufgabe 2.1: Schreibe Funktionen zur Vervierfachung, Verachtfachung, Versechzehnfachung ihrer Eingabewerte, ohne den "Malpunkt" * zu benutzen.

2.1 Variable

Gelegentlich möchte man das Ergebnis eines Logo-Aufrufes "konservieren". Man kann dies tun, indem man den Wert in einer Variablen abspeichert; etwa so:

```
SETZE  "X  VERDOPPLUNG  17                                  (*)
```

Den Wert dieser Variablen kann man nun folgendermaßen abrufen:

```
WERT  "X                                                    (**)
ERGEBNIS:  34
```

An Stelle von WERT "X kann man auch die Kurzform :X verwenden:

```
:X
ERGEBNIS: 34
```

Merke: Logo unterscheidet zwischen dem **Namen** und dem **Wert** von Variablen. Als Variablenname kann fast jede Zeichenkette vorkommen (etwa: A, B5, XYZ, HUGO, ZINSSATZ, C128, ...). Nehmen wir zum Beispiel X als Variablennamen. Dann führt der Logo-Aufruf "X stets dazu, daß uns Logo den Namen von X, also X selbst zurückgibt, während der Aufruf WERT "X beziehungsweise :X stets den Wert der Variablen des Namens X liefert.

Die Unterscheidung zwischen dem Namen und dem Wert von Variablen gibt es in kaum einer anderen Programmiersprache. Sie trägt entscheidend zur Leistungsfähigkeit von Logo bei.

Hier noch einige Beispiele:

```
SETZE "X  18 + VERDOPPLUNG  19
"X
ERGEBNIS:  X

WERT  "X
ERGEBNIS:  56
:X
ERGEBNIS:  56
```

Durch den folgenden Aufruf wird der Wert der Variablen X um eins erhöht:

```
SETZE  "X   ( WERT  "X ) + 1                    (#)
(bzw.:  SETZE  "X   :X + 1 )                    (##)
```

WERT "X (bzw. :X)
ERGEBNIS: 57

Die Beispiele (*), (**), (#) und (##) kann man so lesen:

(*): Weise der Variablen mit dem Namen X den Wert des Logo- Ausdrucks VERDOPPLUNG 17 zu.

(**): Gib den Wert der Variablen mit dem Namen X zurück.

(#) und (##): Weise der Variablen mit dem Namen X den um eins erhöhten Wert der Variablen desselben Namens zu. (Kurz: Erhöhe den Wert der Variablen mit dem Namen X um eins).

2.2 Wiederholungen

Die Funktion VERDOPPLUNG tut genau das, was sie soll: sie liefert als Ausgabewert das Doppelte des Eingabewertes; eine Eingabe hat eine Ausgabe zur Folge. Wenn wir die gesamte Tabelle 1.1 ausgedruckt haben wollen, dann müssen wir die Verdopplungsfunktion mehrmals hintereinander aufrufen, die Ergebnisse jeweils ausdrucken und zur Weiterverarbeitung abspeichern.

Es gibt mehrere grundlegend verschiedene Möglichkeiten, um Logo-Aufrufe (oder Aufrufketten) zu wiederholen. Die wichtigsten davon sollen in diesem Kapitel am Beispiel des Problems *"Schachbrett-Tabelle"* behandelt werden (vergleiche Tabelle 1.1).

Zunächst zum WIEDERHOLE (kurz WH) - Befehl von Logo. Wir können ihn im Direktbetrieb ausprobieren. Das, was wiederholt werden soll, muß in eckige Klammern geschrieben werden. Die hinter dem Grundwort WIEDERHOLE (bzw. WH) stehende Zahl gibt an, wie oft es zu wiederholen ist.

```
SETZE "W "OTTO
WH 4 [ DZ :W  SETZE "W OL :W ]
OTTO
OTT
OT
O
```

```
SETZE "X 1
WH 64 [ DZ :X  SETZE "X  VERDOPPLUNG :X ]
1
2
4
8
16
32
64
128
...
9.22338E18
```

Aufgabe 2.2: Schreibe die letzte Zahl als "richtige" Dezimalzahl.

Da es mühsam ist, längere Aufrufe immer wieder im Direktbetrieb einzutippen, schreiben wir uns einen neuen Befehl dafür. Damit die Prozedur besser lesbar wird, sind die langen Namen verwendet worden. Wir starten mit LERNE SCHACHTABELLE (oder kurz: PR SCHACHTABELLE):

```
PR SCHACHTABELLE
 SETZE  "X  1
 WIEDERHOLE 64 [ DRUCKEZEILE :X SETZE "X VERDOPPLUNG :X ]
ENDE
```

Mit Hilfe der RUN/STOP-Taste verlassen wir den Editor. Logo meldet: SCHACHTABELLE GELERNT. Also probieren wir es aus: der Aufruf SCHACHTABELLE führt tatsächlich zu der oben im Direktbetrieb produzierten Tabelle.

2.3 Sprünge

Wenn man das Logo Grundwort WIEDERHOLE verwendet, dann muß man schon vor Eintritt in die Wiederholungsschleife die Anzahl der Wiederholungen kennen. Dies ist zwar bei der Schachtabelle der Fall; es gibt aber auch Situationen, wo sich erst in der Schleife entscheidet, ob man mit der "Schleifenarbeit" fertig ist oder nicht. Deshalb wollen wir uns noch zwei weitere Methoden ansehen, mit denen man die Abbruchbedingung innerhalb der Schleife überprüfen kann.

In Logo ist es möglich, daß man in Prozeduren **Markierungen** anbringt, zu denen man später springen kann. Hier die Schachtabelle mit Hilfe des Sprungbefehls GEHE:

```
 1:  PR SCHACHTABELLE.SPRUNG
 2:   SETZE  "ZAEHLER  1
 3:   SETZE  "X  1
 4:   MARKIERUNG1:
 5:   DRUCKEZEILE  :X
 6:   SETZE  "ZAEHLER  :ZAEHLER + 1
 7:   SETZE  "X   VERDOPPLUNG  :X
 8:   WENN  :ZAEHLER > 64   DANN   RUECKKEHR
 9:   GEHE  "MARKIERUNG1
10:  ENDE
```

Erläuterungen zur Prozedur SCHACHTABELLE.SPRUNG: Zunächst sei daran erinnert, daß die Zeilennummern nicht zur Prozedur selbst gehören. Der lange Prozedurname ist zwar etwas umständlich zu tippen, aber er zeigt gleich, worum es geht. Zur Kontrolle darüber, mit welchem Feld wir es gerade zu tun haben, wurde in Zeile 2 die Hilfsvariable ZAEHLER eingeführt. Die Variable X gibt die Anzahl der Körner wieder. In Zeile 4 wird durch den Namen MARKIERUNG1 eine Markierung in der Prozedur gesetzt. Markierungsnamen sind (fast) beliebig wählbar. A0, X15, XAVER und SCHLEIFENANFANG sind zum Beispiel alles erlaubte Markierungsnamen. Die Definition einer Markierung wird durch einen Doppelpunkt abgeschlossen, der unmittelbar hinter dem Markierungsnamen steht. Der Doppelpunkt gehört aber nicht mehr zum Markierungsnamen. Der Prozedurteil von Zeile 4 bis Zeile 9 wird als *Schleife* bezeichnet. Der Befehl GEHE "MARKIERUNG1 bewirkt den Rücksprung zur Stelle MARKIERUNG1. In Zeile 6 und 7 werden die Variablenwerte jeweils dem nächsten Feld angepaßt. Der Befehl RUECKKEHR in Zeile 8 bewirkt, daß Logo an die Stelle zurückkehrt, von der aus die Prozedur aufgerufen wurde. Dies kann die Rückkehr zu einer anderen Prozedur oder auch, wie im obigen Beispiel, zum Direktbetrieb zur Folge haben.

Wir können uns den Ablauf der Prozedur anschaulich folgendermaßen vorstellen (vergleiche auch mit Bild 1.2):

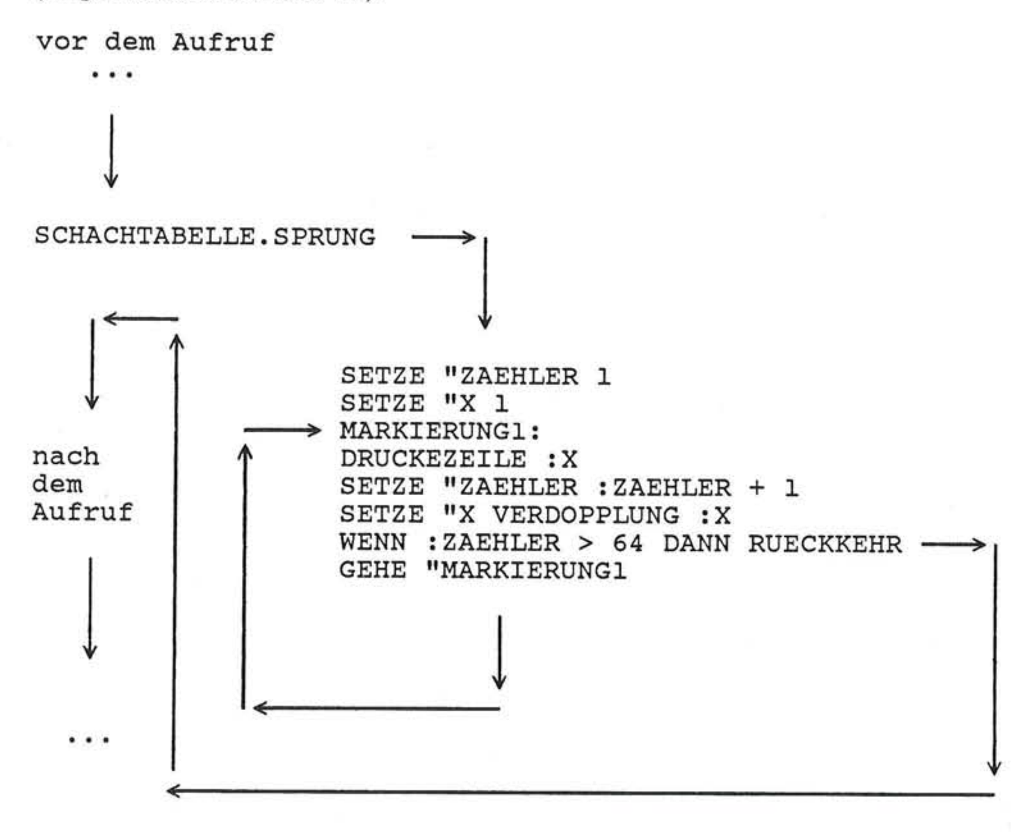


Bild 2.1: Ablaufdiagramm von SCHACHTABELLE.SPRUNG

Aufgabe 2.3: Was würde passieren, wenn in Zeile 4 hinter dem Wort MARKIERUNG1 der Doppelpunkt fehlen würde? Probiere es aus!

Übrigens: Wenn du den langen Namen SCHACHTABLLE.SPRUNG nicht magst, kannst du dir auch eine kleine Prozedur mit einem Kurznamen zum Aufruf schreiben; zum Beispiel so:

```
PR STS
 SCHACHTABELLE.SPRUNG
ENDE
```

Gegen kleine, überschaubare Sprünge in Prozeduren ist wenig einzuwenden. In manchen Programmiersprachen (wie zum Beispiel BASIC) hat der Benutzer außer dem Spungbefehl praktisch keine weitere Möglichkeit, den Kontrollfluß des Programmes in andere Bahnen zu lenken. In solchen Sprachen muß dann sehr viel gesprungen werden, und das macht die Programme unübersichtlich und fehleranfällig.

Mit der im nächsten Abschnitt beschiebenen Methode kann man praktisch vollständig auf Sprünge verzichten.

2.4 Prozeduren, die sich selbst aufrufen

In der Programmiersprache Logo ist es möglich, daß sich Prozeduren selbst aufrufen. Man bezeichnet dieses Verfahren als **Rekursion**. Wir schauen uns dies am besten wieder am Beispiel der Schachtabelle an.

```
PR SCHACHTABELLE.ER
 SETZE  "ZAEHLER  1
 SETZE  "X  1
 SCHACHTABELLE.ER.HP
ENDE

PR SCHACHTABELLE.ER.HP
 DRUCKEZEILE  :X
 SETZE  "ZAEHLER  :ZAEHLER + 1
 SETZE  "X  VERDOPPLUNG  :X
 WENN  :ZAEHLER > 64  DANN  RUECKKEHR
 SCHACHTABELLE.ER.HP
ENDE
```

Erläuterungen zur Prozedur SCHACHTABELLE.ER: Die Buchstaben ER im Prozedurnamen werden etwas weiter unten erklärt. Die Prozedur SCHACHTABELLE.ER dient nur dazu, die Variablen ZAEHLER und X auf die richtigen Anfangswerte zu setzen und danach das eigentliche "Arbeitspferd" aufzurufen, nämlich die Prozedur SCHACHTABELLE.ER.HP. (Die Buchstaben HP stehen dabei für "Hilfsprozedur"). Die ersten Zeilen der Hilfsprozedur entsprechen genau dem Schleifenbeginn in der Prozedur SCHACHTABELLE.SPRUNG. Die letzte Aktion der Prozedur SCHACHTABELLE.ER.HP besteht darin, daß sie sich selbst aufruft.

Prozeduren (bzw. Funktionen), die sich selbst aufrufen, werden **rekursiv** genannt. Der Selbstaufruf kann dabei an verschiedenen Stellen erfolgen. Wenn jedoch, wie im Beispiel unserer obigen Hilfsprozedur, der rekursive Aufruf nur an einer einzigen Stelle steht und wenn er darüberhinaus die letzte Aktion der Prozedur darstellt, dann liegt ein sehr wichtiger Sonderfall der Rekursion vor. Solche Prozeduren heißen **endrekursiv**. Die Buchstaben ER im Prozedurnamen sollen andeuten, daß es sich um eine endrekursive Lösung handelt. Wir werden auch später diese Kennzeichnung bei endrekursiven Prozeduren anbringen.

Merke: Logo-Funktionen können nie endrekursiv sein, denn die letzte Aktion einer Funktion ist immer die Rückgabe des Funktionswertes (also nie der Selbstaufruf). Beachte auch das Beispiel der Prozedur KOERNERZAHL in Abschnitt 2.6.

Bei allen drei Versionen der Prozedur SCHACHTABELLE wurde immer nur die Anzahl der Reiskörner pro Feld ausgedruckt. Es wäre sehr viel informativer, wenn vor dieser Zahl die jeweilige Feldnummer stünde.

Aufgabe 2.4: Ändere die Schachtabelle-Prozeduren so ab, daß in jeder Zeile zuerst die Feldnummer und dann die Körnerzahl gedruckt wird.

2.5 Lokale Variable

Die bisher besprochenen Versionen der Prozedur SCHACHTABELLE sind nur als erste Versuche anzusehen, die noch in vielerlei Hinsicht verbesserungsfähig sind. Im folgenden werden nun einige dieser Verbesserungen diskutiert. Dies soll am Beispiel der Prozedur SCHACHTABELLE.SPRUNG geschehen; die Änderungen sind dann leicht auf die beiden anderen Prozeduren (SCHACHTABELLE und SCHACHTABELLE.ER) zu übertragen.

Eine Variable ist für Logo ein Name, der einen Wert besitzt. Mit dem Kommando ZEIGE NAMEN (kurz ZG NAMEN) kann man sich alle Namen zeigen lassen, die Logo zu diesem Zeitpunkt kennt. Probieren wir es gleich einmal aus:

```
ZEIGE NAMEN
"X .. 1.84467E19      (lies: der Wert von X ist 1.84467E19)
"ZAEHLER .. 65        (lies: der Wert von ZAEHLER ist 65)
```

Logo hatte also in der Prozedur SCHACHTABELLE.SPRUNG schon die Werte für das 65-te Feld berechnet und in seinem Arbeitsspeicher abgelegt; aber vor dem Ausdrucken dieser Werte war die Abbruchbedingung erreicht worden. Dennoch kann man die Variablen X und ZAEHLER auch nach Ende des Laufs der Prozedur SCHACHTABELLE.SPRUNG benutzen; also zum Beispiel ihre Werte abfragen, sie mit neuen Werten belegen, usw. Variable, zu denen man auf diese Weise im Direktbetrieb Zugriff hat, heißen **globale Variable**.

Es mag im Augenblick als ganz schön erscheinen, daß man sich die neuen Werte der globalen Variablen X und ZAEHLER noch im Direktbetrieb anschauen kann. Wenn man aber größere Programme schreibt, wird man sehr bald erkennen, daß es höchst fehleranfällig ist, wenn man mit vielen globalen Variablen arbeitet. Jede Prozedur sollte möglichst genau das tun, was ihr Prozedurname besagt. Von der Prozedur SCHACHTABELLE.SPRUNG erwarten wir, daß sie die entsprechende Tabelle ausdruckt. Es gehört aber nicht mehr zu ihrem "Geschäft", daß sie nach ihrem Ablauf noch irgendwelche Speicherzellen des Computers in Beschlag nimmt. Wenn sie während ihres Laufs gewisse Hilfsvariable benötigt, dann ist das in Ordnung, und sie bekommt sie auch zur Verfügung gestellt, aber nach ihrem Lauf sollte wieder "reiner Tisch" gemacht werden. Dieses Problem betrifft nicht nur Logo, sondern jede Programmiersprache. In jeder vernünftigen Sprache gibt es die folgende Standardlösung dafür: man verhindert, daß die Variablen X und ZAEHLER als globale Variablen erzeugt werden, indem man sie in der Prozedur SCHACHTABELLE als **lokale Variable** deklariert. Dies geschieht mit Hilfe des Logo-Grundworts LOKAL. Lokale Variable "leben" nur so lange, wie die Prozedur läuft, in der sie definiert sind. Nach Ablauf dieser Prozedur sind sie wieder ver-

schwunden, und der von ihnen belegte Speicherplatz wird wieder frei gemacht. Hier die verbesserte Prozedur (der Zusatz "V2" steht für Version 2):

```
PR SCHACHTABELLE.SPRUNG.V2
 LOKAL  "ZAEHLER
 SETZE  "ZAEHLER  1
 LOKAL  "X
 SETZE  "X  1
 MARKIERUNG1:
 ( DRUCKEZEILE  :ZAEHLER  :X )
 SETZE  "ZAEHLER  :ZAEHLER + 1
 SETZE  "X   VERDOPPLUNG  :X
 WENN  :ZAEHLER > 64   DANN RUECKKEHR
 GEHE  "MARKIERUNG1
ENDE
```

Probiere jetzt folgendes im Direktbetrieb aus:

```
ZEIGE NAMEN
"X .. 1.84467E19
"ZAEHLER .. 65
?<Blinker>
```

```
VERGISS NAMEN    (Die globalen Variablen werden gelöscht).
ZEIGE NAMEN
?<Blinker>
```

Es erscheinen nur noch das Logo-Bereitschaftszeichen (in Form eines Fragezeichens) und der Blinker. Logo kennt keine globalen Variablen mehr. Wir experimentieren weiter:

```
SCHACHTABELLE.SPRUNG.V2
1 1
2 2
3 4
 ...
```

```
ZEIGE NAMEN
?<Blinker>
```

Auch jetzt bringt Logo nur das Bereitschaftszeichen und den Blinker; durch den Lauf von SCHACHTABELLE.SPRUNG.V2 wurden keine globalen Variablen erzeugt.

Schließlich wollen wir noch eine weitere Verbesserung durchführen. Der Bildschirm faßt nur 25 Zeilen. Nach dem Lauf der Prozedur SCHACHTABELLE sind also nur noch die Ergebnisse der Felder 41 bis 64 zu sehen. Es wäre doch wünschenswert, daß wir beim Aufruf der Prozedur SCHACHTABELLE an Stelle der festen Felderzahl 64 eine variable Obergrenze eingeben könnten, bei der die Prozedur dann ihren Lauf stoppt. Wie man Prozeduren mit Eingabewerten "bestückt", haben wir schon im vorigen Kapitel gesehen. Ganz entsprechend geht es auch hier. Abschließend eine einigermaßen komplette Version der Prozedur SCHACHTABELLE:

```
PR SCHACHTABELLE.SPRUNG.V3   :OBERGRENZE
 LOKAL  "ZAEHLER
 SETZE  "ZAEHLER  1
 LOKAL  "X
 SETZE  "X  1
 MARKIERUNG1:
 ( DRUCKEZEILE  :ZAEHLER  :X )
 SETZE  "ZAEHLER  :ZAEHLER + 1
 SETZE  "X   VERDOPPLUNG  :X
 WENN  :ZAEHLER > :OBERGRENZE  DANN RUECKKEHR
 GEHE  "MARKIERUNG1
ENDE
```

Schließlich noch ein kleiner Abschlußtest im Direktbetrieb:

```
VERGISS NAMEN
SCHACHTABELLE.SPRUNG.V3  8
1 1
2 2
3 4
4 8
5 16
6 32
7 64
8 128
ZEIGE NAMEN
?<Blinker>
```

Logo kennt auch nach diesem Lauf keine globalen Namen. Dieses Beispiel zeigt, daß auch die Eingabeparameter von Prozeduren (im obigen Beispiel der Parameter OBERGRENZE) lokale Variable sind.

Aufgabe 2.5: Schreibe eine Prozedur STS3 als *Kurzaufruf* für SCHACHTABELLE.SPRUNG.V3.

Aufgabe 2.6: Übertrage die in diesem Abschnitt besprochenen Verbesserungen sinngemäß auch auf die Prozeduren SCHACHTABELLE und SCHACHTABELLE.ER.

Der letzte Aufruf hat uns gezeigt, daß Variable, die als Eingabeparameter von Prozeduren vorkommen, stets lokal sind. Dies bedeutet, daß wir die rekursive Version von SCHACHTABELLE auch folgendermaßen schreiben können:

```
PR SCHACHTABELLE.ER.V2  :OG
 SCHACHTABELLE.ER.V2.HP  :OG  1  1
ENDE

PR SCHACHTABELLE.ER.V2.HP  :OG  :Z  :X
 WENN :Z > :OG  DANN  RUECKKEHR
 ( DRUCKEZEILE  :Z  :X )
 SCHACHTABELLE.ER.V2.HP  :OG  (:Z+1)  (VERDOPPLUNG :X)
ENDE
```

Erläuterungen zur Prozedur SCHACHTABELLE.ER.V2: Die Buchstaben ER, V2 und HP sollen wieder für "endrekursiv", "Version 2" und "Hilfsprozedur" stehen. Die Variablen OG und Z stehen für Obergrenze und Zähler, der Wert von X ist nach wie vor die Anzahl der Reiskörner auf dem jeweiligen Feld.

Der entscheidende Unterschied zur letzten endrekursiven Version ist, daß die Hilfsprozedur jetzt über Eingabeparameter verfügt. Wenn dies der Fall ist, dann braucht man Variable nicht mehr wie vorher mit dem SETZE-Befehl zu verändern, sondern man weist ihnen beim Aufruf einfach den richtigen Wert zu. Die Zeilen:

```
SETZE "ZAEHLER 1
SETZE "X 1
```

in der alten Version von SCHACHTABELLE.ER dienten ja nur dazu, die in der Hilfsprozedur benötigten Variablen mit den richtigen Anfangswerten zu bestücken. Dies geht aber viel einfacher, indem wir diese Variablen zu Eingabeparametern der Hilfsprozedur machen und die Hilfsprozedur dann mit den richtigen Werten für diese Parameter aufrufen. Diesem Ziel dient der Aufruf

```
SCHACHTABELLE.ER.V2.HP :OG 1 1
```

Durch diesen Aufruf erhalten die Variablen Z und X (in dieser Reihenfolge) jeweils den Wert 1 zugeordnet. Nachdem der DRUCKEZEILE-Befehl abgearbeitet worden ist, ist die Zeile

```
SCHACHTABELLE.ER.V2.HP :OG (:Z+1) (VERDOPPLUNG :X)
```

zu verarbeiten. Dazu müssen zuerst die neuen Werte für die Eingabeparameter ermittelt werden. Der Parameter OG verändert sich nicht, der augenblickliche Wert von Z ist 1, damit erhält Z als neuen Wert :Z+1, also 2. Der neue Wert von X ist das Ergebnis des Aufrufes VERDOPPLUNG 1, also ebenfalls 2. Beim nächsten Aufruf erhält Z den Wert 3 und X den Wert 4, danach Z den Wert 4 und X den Wert 8, usw.

Man kann den Ablauf von Prozeduren sehr gut im **Protokollmodus** verfolgen. Durch PE (für PROTOKOLL EIN) und PA (für PROTOKOLL AUS) läßt sich dieser Modus ein- und ausschalten. Der Protokollmodus eignet sich besonders gut zum Verständnis rekursiver Prozeduren. Wir probieren es am besten einfach aus:

```
PE
SCHACHTABELLE.ER.V2 5
```

Im Protokollmodus druckt Logo jede Prozedur-Zeile vor ihrer Ausführung am Bildschirm und wartet. Erst nach Eingabe eines <RETURN>-Zeichens fährt Logo mit der Abarbeitung der ausgedruckten Zeile fort. Gib nach Ablauf des Protokolls das Kommando PA (Protokoll aus) ein.

Aufgabe 2.7: Durch die Verwendung ziemlich langer Namen wurde das Protokoll etwas unübersichtlich. Schreibe kurznamige Prozeduren STER2 und STER2H an Stelle von SCHACHTABELLE.ER.V2 und SCHACHTABELLE.ER.V2.HP und laß diese neuen Prozeduren im Protokollmodus laufen.

Aufgabe 2.8: Ändere die SCHACHTABELLE-Prozeduren so ab, daß für jedes Feld zugleich die laufende Summe aller Körner bis zu diesem Feld berechnet und ausgedruckt wird (abgesehen von der stellengerechten Darstellung also etwa folgendermaßen):

```
1     1     1
2     2     3
3     4     7
4     8    15
5    16    31
6    32    63
7    64   127
8   128   255
    ...
```

Tabelle 2.1

Vergleiche die Zahlen in der "Summenspalte" (3. Spalte) mit den Zahlen in der "Feldspalte" (2. Spalte) der jeweils nächsten Zeile. Formuliere eine Gesetzmäßigkeit und begründe sie.

Eine Bemerkung zum stellengerechten Ausdrucken: Stellengerechte Ausdrucke sind auf den heutigen Mikrocomputern meist nur mit Mühe herzustellen. Der besseren Lesbarkeit halber sind die Ausdrucke in diesem Buch dennoch in stellengerechter Form gegeben, auch wenn sie bei den meisten Prozeduren auf dem Bildschirm nicht stellengerecht erscheinen. Eine Möglichkeit, um stellengerechte Ausdrucke durch eigene Prozeduren zu erzeugen, ist in dem Buch *"Programmieren lernen mit Logo"* (Jochen Ziegenbalg), Hanser Verlag, Kapitel 9, beschrieben.

2.6 Rekursive Funktionen

Weiter oben mußten wir feststellen, daß Funktionen (leider) nie endrekursiv sind. Hierzu ein Beispiel. Der Aufruf KOERNERZAHL :N soll nur die Anzahl der Körner auf dem Feld Nr. :N als Funktionswert zurückgeben und ansonsten keine Nebenwirkungen haben.

```
PR KOERNERZAHL  :N
 WENN  :N = 1  DANN  RUECKGABE  1
 RUECKGABE  2 * KOERNERZAHL (:N - 1)
ENDE
```

Einige Aufrufe:

KOERNERZAHL 1
ERGEBNIS: 1

KOERNERZAHL 5
ERGEBNIS: 16

KOERNERZAHL 20
ERGEBNIS: 524288

KOENERZAHL 30
ERGEBNIS: 536870912

KOERNERZAHL 100
SPEICHER VOLL! ...

Um den Ablauf der Prozedur KOERNERZAHL besser zu verstehen, schauen wir sie uns im Protokollmodus an:

```
PE
KOERNERZAHL 3
AUFRUF--- KOERNERZAHL 3
  WENN :N = 1 DANN RUECKGABE 1
  RUECKGABE 2 * KOERNERZAHL (:N - 1)
 AUFRUF--- KOERNERZAHL 2
   WENN :N = 1 DANN RUECKGABE 1
   RUECKGABE 2 * KOERNERZAHL (:N - 1)
  AUFRUF--- KOERNERZAHL 1
    WENN :N = 1 DANN RUECKGABE 1
   RUECKG. 1
  FERTIG KOERNERZAHL
  RUECKG. 2
 FERTIG KOERNERZAHL
 RUECKG. 4
FERTIG KOERNERZAHL
ERGEBNIS:  4
```

Merke: Logo leitet Funktionswerte mit dem RUECKGABE-Befehl immer an die Stelle weiter, von wo aus der Aufruf erfolgte. Dies kann insbesondere auch ein früherer Aufruf derselben Funktion sein. Genau dies ist beim Aufruf der Funktion KOERNERZAHL der Fall.

Schauen wir uns einmal die *mathematische Substanz* eines solchen Protokolls an:

```
KOERNERZAHL 4   =   2 * KOERNERZAHL 3
                =   2 * ( 2 * KOERNERZAHL 2 )
                =   2 * ( 2 * ( 2 * KOERNERZAHL 1 ) )
                =   2 * ( 2 * ( 2 * 1 ) )
                =   2 * ( 2 * 2 )
                =   2 * 4
                =   8
```

Die Funktion KOERNERZAHL ruft sich im letzten Beispiel dreimal selbst auf. Logo muß dabei genau Buch führen, wohin der Funktionswert jeweils zurückzugeben ist und wie er weiterzuverarbeiten ist. Dazu legt Logo einen Speicher an, den sogenannten *Rekursionsstack*. Bei jedem Aufruf kommen die Angaben zur Weiterverarbeitung des Funktionswertes auf den Rekursionsstack; bei jeder Rückgabe werden sie nach dem Prinzip **last in - first out** wieder vom Stack genommen. Je tiefer die Rekursion geht, um so stärker wird der Speicher belastet. Wenn man den Eingabewert :N nur hinreichend groß wählt, kann man den Speicher sprengen. Im obigen Beispiel führte deshalb der Aufruf KOERNERZAHL 100 zur Fehlermeldung SPEICHER VOLL!.

Dies ist bedauerlich, weil die obige Version der Funktion Körnerzahl von allen denkbaren Lösungen sprachlich sicher die eleganteste ist. Aber was hilft's? Prozeduren oder Funktionen, bei deren Benutzung sehr bald der Speicher voll läuft, sind nur von theoretischer Bedeutung.

Deshalb sei im folgenden noch eine Standard-Methode gezeigt, mit der man voll rekursive Funktionen, wie die obige, in endrekursive überführen und damit das Vollaufen des Speichers vermeiden kann.

```
PR KOERNERZAHL.ER  :N
 LOKAL  "ANZAHL
 SETZE  "ANZAHL  1
 KOERNERZAHL.ER.HP  :N  1
 RUECKGABE · :ANZAHL
ENDE

PR KOERNERZAHL.ER.HP  :N  :I
 WENN  :I = :N  DANN RUECKKEHR
 SETZE  "ANZAHL  2 * :ANZAHL
 KOERNERZAHL.ER.HP  :N  (:I + 1)
ENDE
```

KOERNERZAHL.ER 64
ERGEBINS: 9.22338E18

KOERNERZAHL.ER 100
ERGEBNIS: 6.33826E29

Aufgabe 2.9: Stelle das letzte Ergebnis als normale Dezimalzahl dar.

Wenn wir die Prozedur KOERNERZAHL.ER.HP im Direktbetrieb aufrufen, etwa durch KOERNERZAHL.ER.HP 20 1, dann bringt Logo die Fehlermeldung, daß der Name ANZAHL unbekannt ist. Wieso hat Logo vorhin den Namen Anzahl gekannt, als die Hilfsprozedur von der Prozedur KOERNERZAHL.ER aus aufgerufen wurde? Nun, das liegt an der folgenden sehr wichtigen Tatsache:

Merke: Aufgerufene Prozeduren übernehmen alle Variablen (einschließlich ihrer Werte) von der aufrufenden Prozedur.

Aufgabe 2.10: Schreibe eine "Sprung"-Version der Funktion KOERNERZAHL.

Aufgabe 2.11: Bereinige den Arbeitsspeicher und speichere die verbleibenden Prozeduren unter einem geeigneten Dateinamen (zum Beispiel: SCHACH) ab. Hinweise dazu sind in Abschnitt 1.12 zu finden.

Die Version KOERNERZAHL.ER ist auch noch eine relativ "saubere" Lösung: sie schafft keine globalen Variablen, die später im Speicher "herumvagabundieren", und sie gibt nach ihrem Lauf den benötigten Speicher wieder frei. Das einzige, was sie tut, ist das, was von ihr erwartet wird, nämlich den gesuchten Funktionswert zurückzugeben. Ansonsten hat sie keine Nebenwirkungen. Dies wird für uns im folgenden die Standardmethode sein, um Funktionen zu schreiben, die Wiederholungsschleifen enthalten.

2.7 Listen

Die obigen Versionen der Prozedur SCHACHTABELLE haben einen Schönheitsfehler. Werte, die nur am Bildschirm oder auf Papier ausgedruckt wurden, können nicht mehr weiterverarbeitet werden.

Einen gewissen Ausweg aus diesem Dilemma stellt die Funktion KOERNERZAHL dar. Aber sie liefert immer nur eine einzige Zahl als Funktionswert. Wenn man zum Beispiel Gesetzmäßigkeiten in der Entwicklung der Körnerzahlen per Programm untersuchen will, dann benötigt man unter Umständen viele Werte "auf einen Schlag".

Im folgenden werden wir eine Möglichkeit kennenlernen, die Funktionswerte so zu speichern, daß man sie geschickt weiterverarbeiten kann. Wir werden die Körnerzahlen in einer "Liste" abspeichern. Eine **Liste** ist ein Speicherelement (vornehm ausgedrückt: ein Datentyp), in dem man Zahlen, Wörter (Zeichenketten) und sogar wieder Listen abspeichern kann. Die einzelnen Elemente einer Liste werden, durch Leerzeichen getrennt, nebeneinander geschrieben. Anfang und Ende jeder Liste sind durch eckige Klammern gekennzeichnet. Hier einige Beispiele für Listen:

```
[ 1  2  3  4  5 ]                    ( 5 Elemente)
[ A1  5  B2  7  C5  8 ]              ( 6 Elemente)
```

Die folgende Liste besteht aus 8 Elementen; den *Wörtern* A, B, C, D, der *Teilliste* [E F] sowie den *Wörtern* G, H und K: [A B C D [E F] G H K].

Die leere Liste [ ] enthält überhaupt kein Element.

Die Listenverarbeitung ist eines der wesentlichen Merkmale der Programmiersprache Logo. Logo ist ein Dialekt der Sprache **Lisp**, deren Name für **List Processing Language** (deutsch: listenverarbeitende Sprache) steht.

Es gibt eine Vielzahl von Logo-Grundbefehlen, mit denen man Listen auseinandernehmen und zusammenbauen kann. Manche dieser Befehle entsprechen Grundbefehlen, die wir schon im Zusammenhang mit der Verarbeitung von Wörtern kennengelernt haben. Hier einige Beispiele:

ERSTES [X Y Z]   (Kurzform: ER)
ERGEBNIS: X

OHNEERSTES [A1 B2 C3 D4]   (Kurzform: OE)
ERGEBNIS: [B2 C3 D4]

LETZTES [JAN FEB MRZ APR]   (Kurzform: LZ)
ERGEBNIS: APR

OHNELETZTES [[1 2] [3 4] [5 6]] (Kurzform: OL)
ERGEBNIS: [[1 2] [3 4]]

Dem Grundbefehl WORT entspricht bei der Listenverarbeitung der Grundbefehl SATZ. Mit diesem Grundbefehl kann man Listen zusammenfügen.

SATZ [A B] [C D E]
ERGEBNIS: [A B C D E]

Alle diese Befehle sind auch auf *geschachtelte* Listen anwendbar und in Verbindung mit Variablen möglich; hier einige Beispiele:

ERSTES [[X Y Z] D E F]
ERGEBNIS: [X Y Z]

OHNELETZTES [[U V W] [A [B [C]]]]
ERGEBNIS: [[U V W]]

SETZE "L1 [1 2 3 4]
SATZ :L1 [X Y Z]
ERGEBNIS: [1 2 3 4 X Y Z]

Der Grundbefehl LISTE fügt seine beiden Eingaben zu einer Liste zusammen.

LISTE "A "B
ERGEBNIS: [A B]

LISTE [1 2] [C D]
ERGEBNIS: [[1 2] [C D]]

Wenn man mehr als zwei oder nur ein Element zu einer Liste zusammenfügen lassen will, muß man (ähnlich wie bei den Druckbefehlen) den gesamten Aufruf in runde Klammern schreiben.

```
(LISTE  "X )
ERGEBNIS:  [ X ]

(LISTE  "A  [ B C ]  "D )
ERGEBNIS:  [ A  [ B C ]  D ]
```

Bei den letzten beiden Beispielen sind die Leerzeichen zwischen X bzw. D und der schließenden (runden) Klammer sehr wichtig. Wenn sie fehlten, würde Logo die Zeichenkette X) bzw. D) jeweils als zweielementige Zeichenketten ansehen und die schließenden Klammern vermissen!

Auch das Grundwort SATZ "verkraftet" Mehrfacheingaben in Verbindung mit runden Klammern.

Aufgabe 2.12: Mache dir die Wirkungsweise von LISTE und SATZ klar, indem du bei den obigen Beispielen an Stelle von LISTE jeweils SATZ eingibst und umgekehrt.

Zwei weitere wichtige Grundwörter zur Listenverarbeitung sind MITERSTEM (kurz ME) und MITLETZTEM (kurz ML). Auch dazu einige Beispiele:

MITERSTEM "H [A U S]
ERGEBNIS: [H A U S]

MITLETZTEM "OMEGA [ALPHA]
ERGEBNIS: [ALPHA OMEGA]

MITERSTEM [A 1] [[B 2] [C 3]]
ERGEBNIS: [[A 1] [B 2] [C 3]]

Die Grundbefehle MITERSTEM bzw. MITLETZTEM erwarten zwei Eingaben: ein beliebiges Objekt und eine Liste. Das Objekt kann zum Beispiel eine Zahl, ein Wort oder selbst eine Liste sein. Es wird durch den Grundbefehl als erstes (bzw. letztes) Element in die Liste eingefügt.

Wir können uns nun dem Ausgangsproblem dieses Abschnitts zuwenden, die gesamte Schachtabelle "auf einen Schlag" in Form einer Liste auszugeben. Die entsprechende Funktion soll nun sinngemäß SCHACHLISTE heißen. Sie entsteht durch Abwandlung der Prozedur SCHACHTABELLE.SPRUNG.V3 aus Abschnitt 2.5.

```
 1:  PR SCHACHLISTE.SPRUNG  :OBERGRENZE
 2:   LOKAL  "ZAEHLER
 3:   SETZE  "ZAEHLER  1
 4:   LOKAL  "X
 5:   SETZE  "X  1
 6:   LOKAL  "KOERNERLISTE
 7:   SETZE  "KOERNERLISTE  [ ]
 8:   MARKIERUNG1:
 9:   SETZE  "KOERNERLISTE MITLETZTEM  :X  :KOERNERLISTE
10:   SETZE  "ZAEHLER  :ZAEHLER + 1
11:   SETZE  "X  VERDOPPLUNG  :X
12:   WENN :ZAEHLER > :OBERGRENZE RUECKGABE :KOERNERLISTE
13:   GEHE  "MARKIERUNG1
14:  ENDE
```

SCHACHLISTE.SPRUNG 12
ERGEBNIS: [1 2 4 8 16 32 64 128 256 512 1024 2048]

Diese Funktion unterscheidet sich von der Prozedur SCHACHTABELLE.SPRUNG.-V3 in den folgenden drei Punkten: zunächst einmal wurde in Zeile 6 und 7 die lokale Variable KOERNERLISTE eingeführt und mit dem Anfangswert [ ] (leere Liste) belegt. Der frühere Druckbefehl wurde gestrichen; statt dessen wird jetzt die KOERNERLISTE mit dem aktuellen Wert von X als letztem Element bestückt (Zeile 9). Schließlich wird der Lauf der Prozedur nun nicht mehr durch RUECKKEHR, sondern durch die Funktionswertrückgabe beendet (Zeile 12).

Aufgabe 2.13: Füge zwischen Zeile 8 und Zeile 9 den Befehl DRUCKEZEILE :KOERNERLISTE ein und beobachte den Programmlauf. Denke daran, daß der Druckbefehl beim Ausdruck von Listen die äußersten Klammern wegläßt (siehe Abschnitt 1.8). Entferne diesen Druckbefehl danach wieder.

Aufgabe 2.14: Schreibe eine Funktion SCHACHLISTE.ER, die durch Übertragung der Prozedur SCHACHTABELLE.ER.V3 entsteht.

Aufgabe 2.15: Schreibe eine voll rekursive Version der Funktion SCHACHLISTE, die ohne den SETZE-Befehl auskommt.

Aufgabe 2.16: Schreibe eine Funktion SCHACHLISTE.VERBESSERT, welche die folgende aus den Paaren Feldnummer / Körnerzahl bestehende Liste als "Funktionswert" zurückgibt:

[[1 1] [2 2] [3 4] [4 8] [5 16] ...]

2.8 Prüfwörter

Logo verfügt über eine Reihe sehr nützlicher *Prüfwörter*. Dies sind Funktionen, die einen der Werte WAHR oder FALSCH zurückgeben. Die Prüfwörter enden mit einem Fragezeichen.

Mit dem Aufruf ZAHL? :X wird überprüft, ob die Eingabe :X eine Zahl ist oder nicht.

ZAHL? 30031
ERGEBNIS: WAHR

ZAHL? 3.14
ERGEBNIS: WAHR

ZAHL? "ABC
ERGEBNIS: FALSCH (ABC ist ein Wort, keine Zahl)

ZAHL? [6.28]
ERGEBNIS: FALSCH (die Eingabe ist eine Liste)

Mit dem Aufruf WORT? :Y wird überprüft, ob die Eingabe :Y ein Wort ist oder nicht.

WORT? "XAVER
ERGEBNIS: WAHR

WORT? "2.71
ERGEBBIS: WAHR

WORT? [X Y Z]
ERGEBNIS: FALSCH

Die Zahlen werden zu den Wörtern gerechnet:

WORT? 2.71
ERGEBNIS: WAHR

WORT? 1001
ERGEBNIS: WAHR

Mit dem Aufruf LISTE? :Z kann man überprüfen, ob die Eingabe :Z eine Liste ist.

LISTE? [U V W]
ERGEBNIS: WAHR

LISTE? "RST
ERGEBNIS: FALSCH (die Eingabe ist ein Wort)

LISTE? 26.5
ERGEBNIS: FALSCH (die Eingabe ist eine Zahl)

Der Aufruf NAME? :X prüft, ob die Eingabe :X als Name einer Variablen definiert ist. Damit keine Nebenwirkungen auftreten, löschen wir vor dem nächsten Test den Arbeitsspeicher mit Hilfe von ADE.

```
NAME? "X
ERGEBNIS: FALSCH
```

```
SETZE "X 17.8
NAME? "X
ERGEBNIS: WAHR
```

Natürlich kann man beim Arbeiten mit Prüfwörtern auch Variable verwenden. Zum Beispiel:

```
SETZE "Y "HUGO
WORT? WERT "Y
ERGEBNIS: WAHR
```

```
WORT? :Y
ERGEBNIS: WAHR
```

Aufgabe 2.17: Experimentiere auch mit den anderen Prüfwörtern in Verbindung mit Variablen.

Im nächsten Kapitel werden wir sehen, wie man sich selber Prüfwörter schreiben kann, zum Beispiel das Prüfwort TEILT?. Mit dem Aufruf TEILT? :A :B werden wir entscheiden können, ob die Zahl :A die Zahl :B teilt.

Kapitel 3: Aus der Teilbarkeitslehre

3.1 Teiler

Probiere folgendes im Direktbetrieb aus:

DIV 12 2
ERGEBNIS: 6

DIV 15 3
ERGEBNIS: 5

DIV 7 2
ERGEBNIS: 3

DIV 31 7
ERGEBNIS: 4

DIV 3 5
ERGEBNIS: 0

Die Logo-Grundfunktion DIV erwartet zwei Eingabewerte. Sie überprüft, wie oft die zweite Zahl "ganz" in der ersten Zahl "aufgeht", und gibt diese Zahl als Funktionswert zurück. DIV bewirkt offenbar die **Ganzzahldivision**, wie sie schon in der Grundschule geübt wird.

Zur Ganzzahldivision gehört auch die Frage nach dem **Divisionsrest**. Die Logo-Grundfunktion REST liefert uns die entsprechenden Antworten:

REST 12 2
ERGEBNIS: 0

REST 15 3
ERGEBNIS: 0

REST 7 2
ERGEBNIS: 1

REST 31 7
ERGEBNIS: 3

REST 3 5
ERGEBNIS: 3

Mit Hilfe von REST können wir entscheiden, ob eine Zahl eine andere teilt, denn die Zahl a teilt die Zahl b, wenn die Ganzzahldivision von b durch a den Rest Null ergibt. Die Zahl 12 teilt zum Beispiel die Zahl 48, weil "48 geteilt durch 12" den Rest Null liefert.

Aufgabe 3.1: Gib Beispiele für die folgende Aussage an: Die Zahl a teilt die Zahl b genau dann, wenn es eine natürliche Zahl n gibt mit der Eigenschaft: $b = n * a$.

Bei Teilbarkeitsfragen sollte man Divisionen immer mit der Ganzzahldivision (DIV) und nicht mit der Gleitkommadivision (Schrägstrich) ausführen, denn die Gleit-

kommadivision führt häufig zu Ergebnissen, die nur näherungsweise richtig sind (siehe Kapitel 1, Abschnitt 1.3), und dies ist in der Teilbarkeitslehre nicht akzeptabel.

Hier zunächst eine etwas umständliche Möglichkeit, um die Teilbarkeit zu überprüfen:

```
PR TEILT?.UMSTAENDLICH  :A  :B
 WENN (REST  :B  :A) = 0  DANN RUECKGABE "WAHR _____
 _____  SONST RUECKGABE  "FALSCH
ENDE
```

Die **Unterstreichungszeichen** gehören nicht zum Logo-Programm selbst. Sie sollen nur andeuten, daß die dadurch verbundenen *optischen Zeilen* als eine *logische Zeile* gewertet werden. Im Editiermodus sieht die Prozedur TEILT?.UMSTAENDLICH folgendermaßen aus:

```
PR TEILT?.UMSTAENDLICH :A :B
 WENN ( REST :B :A ) = 0 DANN RUECKGABE!
 "WAHR SONST RUECKGABE "FALSCH
ENDE
```

Das Ausrufezeichen am rechten Bildschirmrand gehört nicht zur eigentlichen Prozedur. Es soll nur andeuten, daß die *logische* Programmzeile über den rechten Rand der *optischen* Bildschirmzeile fließt, daß die Anweisung also auf der nächsten Zeile weitergeht. Die automatische Trennung durch das Ausrufezeichen erfolgt manchmal an sehr ungünstigen Stellen. Der besseren Lesbarkeit halber wird in diesem Buch wie im obigen Beispiel das Unterstreichungszeichen verwendet, um anzudeuten, daß zwei (oder mehr) optische Zeilen zusammengehören. Der Prozedurkörper von TEILT?.UMSTAENDLICH besteht also nur aus einer einzigen (logischen) Zeile. Das Unterstreichungszeichen ist beim Eintippen der Prozedur einfach wegzulassen. Wenn die Bildschirmzeile "überläuft", setzt Logo automatisch das Ausrufezeichen.

Weil der RUECKGABE-Befehl bewirkt, daß die Funktion verlassen wird, ist die obige "einzeilige" Version gleichwertig zur folgenden "zweizeiligen" Fassung:

```
PR TEILT?.UMSTAENDLICH.V2  :A  :B
 WENN  (REST :B  :A)  = 0  DANN RUECKGABE  "WAHR
 RUECKGABE  "FALSCH
ENDE
```

In den meisten Programmiersprachen würde man das Problem wohl ähnlich wie im Beispiel TEILT?.UMSTAENDLICH lösen. In Logo geht es aber noch viel besser. Probieren wir zunächst folgendes im Direktbetrieb aus:

```
(REST 6 3) = 0
ERGEBNIS: WAHR

(REST 11 4) = 0
ERGEBNIS: FALSCH
```

Das Gleichheitszeichen liefert die Funktionswerte WAHR bzw. FALSCH, die wir, wie andere Funktionswerte auch, sofort weiterverarbeiten können.

```
PR TEILT?  :A  :B
 RUECKGABE  ( REST  :B  :A )  =  0
ENDE
```

Eine Probe:

TEILT? 4 12
ERGEBNIS: WAHR

TEILT? 5 17
ERGEBNIS: FALSCH

Erläuterung der Funktion TEILT?: Der Ausdruck (REST :A :B) = 0 wird von Logo ausgewertet und ergibt den Wert WAHR oder FALSCH. Dieser Wert wird durch den RUECKGABE-Befehl weitergeleitet und dadurch zum Funktionswert der Funktion TEILT? gemacht. Die Funktion TEILT? hat als Ausgabewert also stets einen der Werte WAHR oder FALSCH. Wir haben derartige Funktionen in Kapitel 2, Abschnitt 2.8 bereits als *Prüfwörter* kennengelernt. So überprüft die Funktion TEILT? zum Beispiel, ob die Zahl :A die Zahl :B teilt. Damit man die Prüfwörter besser erkennen kann, sollte man sie mit einem Fragezeichen versehen. Die Programme werden dadurch besser lesbar.

Wir können nun zum Beispiel alle Teiler einer Zahl ausdrucken:

```
PR TEILER.AUSDRUCK  :N
 LOKAL  "T
 SETZE  "T  1
 SCHLEIFENANFANG:
 WENN  TEILT?  :T  :N   DANN  DRUCKEZEILE  :T
 SETZE  "T  :T + 1
 WENN  :T  >  :N   DANN  RUECKKEHR
 GEHE "SCHLEIFENANFANG
ENDE
```

Einige Aufrufbeispiele:

TEILER.AUSDRUCK 24
1
2
3
4
6
8
12
24

TEILER.AUSDRUCK 18
1
2
3
6
9
18

Erläuterung zur Prozedur TEILER.AUSDRUCK: Diese Prozedur entspricht von der Schleifen-Struktur her völlig der Prozedur SCHACHTABELLE.SPRUNG. T ist der

Name einer Hilfsvariablen, deren Wert von 1 bis :N+1 läuft. Wenn :T ein Teiler von :N ist, wird :T ausgedruckt, sonst nicht. Also werden genau die Teiler von :N ausgedruckt. Zum Vergleich wollen wir noch die entsprechende endrekursive Version betrachten:

```
PR TEILER.AUSDRUCK.ER  :N
 TEILER.AUSDRUCK.ER.HP  :N  1
ENDE

PR TEILER.AUSDRUCK.ER.HP  :N  :T
 WENN  TEILT?  :T  :N   DANN  DRUCKEZEILE :T
 WENN  :T  >  :N  DANN RUECKKEHR
 TEILER.AUSDRUCK.ER.HP  :N  (:T+1)
ENDE
```

Erläuterungen zur Prozedur TEILER.AUSDRUCK.ER: Die Prozedur dieses Namens ruft nur das eigentliche "Arbeitspferd" TEILER.AUSDRUCK.ER.HP auf. (Die Zusätze ER und HP sollen, wie bisher auch, "endrekursiv" und "Hilfsprozedur" bedeuten).

Es ist sehr lehrreich, die Prozedur TEILER.AUSDRUCK.ER im Protokollmodus laufen zu lassen. Hier ein abgekürztes Protokoll, aus dem der Ablauf besonders deutlich hervorgeht:

```
TEILER.AUSDRUCK.ER 6
TEILER.AUSDRUCK.ER.HP 6 1
1
TEILER.AUSDRUCK.ER.HP 6 2
2
TEILER.AUSDRUCK.ER.HP 6 3
3
TEILER.AUSDRUCK.ER.HP 6 4
TEILER.AUSDRUCK.ER.HP 6 5
TEILER.AUSDRUCK.ER.HP 6 6
6
?<BLINKER>
```

Es ist eigentlich unnötig, die Hilfsvariable T alle Werte von 1 bis :N durchlaufen zu lassen. Denn die Teiler treten paarweise auf. Zu jedem **Teiler** gehört sein **Partner** (auch *Komplementärteiler* genannt). Nehmen wir zum Beispiel die Zahl 24. Dann haben wir die folgenden Paare:

Teiler	Partner
1	24
2	12
3	8
4	6

Tabelle 3.1

Aufgabe 3.2: Verbessere die obigen TEILER-Prozeduren so, daß mit jedem Teiler gleich sein Partner ausgedruckt wird. Wie weit muß man dann die Hilfsvariable T noch laufen lassen? Stoppe für verschiedene Werte von :N das Verhältnis von "Laufzeit der langsamen Version" zu "Laufzeit der schnellen Version".

Aufgabe 3.3: Aus Tabelle 3.1 könnte man auf den ersten Blick schließen, daß die Teilerzahl stets gerade ist. Denn die Teiler treten ja immer paarweise auf. Überprüfe dies. Teste die Zahl 1522756.

Unsere bisherigen TEILER-Prozeduren haben noch einen Schönheitsfehler. Werte, die nur am Bildschirm oder auf Papier ausgedruckt wurden, können nicht mehr gut weiterverarbeitet werden. Nehmen wir zum Beispiel an, wir wollen zunächst die Teiler der Zahlen 72 und 148 und danach ihre gemeinsamen Teiler bestimmen. Dann können wir uns mit den obigen Prozeduren zwar die Teiler von 72 und 148 ausdrucken lassen; die gemeinsamen Teiler müßten wir dann aus diesen Listen aber "von Hand" ermitteln. Dies ist natürlich unbefriedigend. Denn wenn wir schon einen Computer haben, dann wollen wir ihn gleich richtig einsetzen und nicht etwa seine Vorarbeit noch von Hand zu überarbeiten haben.

Im folgenden werden wir deshalb die Teiler so abspeichern, daß man sie geschickt weiterverarbeiten kann. Das geeignete Speichermedium ist offenbar die Liste (vergleiche Kapitel 2, Abschnitt 2.7).

Für unsere Teilermengen benötigen wir nur ziemlich einfache Listen. Die Liste [1 2 3 6 9 18] enthält zum Beispiel gerade die Teiler der Zahl 18. Wie wir in Abschnitt 2.7 gesehen haben, kann man auch ganze Listen in einem einzigen Variablennamen verpacken; zum Beispiel:

```
SETZE "L  [ 1 2 3 4 6 8 12 24 ]
WERT "L       (bzw. :L )
ERGEBNIS:  [ 1 2 3 4 6 8 12 24 ]
```

In der Variablen namens L verbergen sich also jetzt gerade die Teiler der Zahl 24. Natürlich hätte man auch einen Namen wählen können, aus dem der Inhalt besser hervorgeht; etwa so:

SETZE "T36 [1 2 3 4 6 9 12 18 36]   (Teiler von 36)

Wir wollen jetzt ein Programm schreiben, mit der wir solche Teilerlisten automatisch herstellen können:

```
PR TEILER  :N
 LOKAL  "L
 SETZE  "L  [ ]
 LOKAL  "T
 SETZE  "T  1
 SCHLEIFENANFANG:
 WENN  TEILT? :T :N  DANN SETZE "L  MITLETZTEM :T :L
 SETZE  "T  :T + 1
 WENN :T > :N  DANN  RUECKGABE :L
 GEHE  "SCHLEIFENANFANG
ENDE
```

TEILER 6
ERGEBNIS: [1 2 3 6]

TEILER 210
ERGEBNIS: [1 2 3 5 7 6 10 14 15 21 30 35 42 70 105 210]

Erläuterungen zur Prozedur TEILER: Der Unterschied zur Prozedur TEILER.AUSDRUCK besteht darin, daß die einzelnen Teiler nicht mehr ausgedruckt,

sondern in der Liste L gespeichert werden. Zunächst wird durch den Befehl SETZE "L [ ] dafür gesorgt, daß die Liste :L leer ist. Durch den Befehl SETZE "L MITLETZTEM :T :L wird der Teiler :T als letztes Element in die Liste :L eingefügt.

Die folgende Tabelle zeigt am Beispiel :N = 6, wie sich die Variablen T und L beim Lauf der Funktion TEILER in den einzelnen Schleifendurchläufen verändern:

Schleifen-durchlauf	Wert von T vorher	nachher	Wert von L vorher	nachher
1	1	2	[ ]	[1]
2	2	3	[1]	[1 2]
3	3	4	[1 2]	[1 2 3]
4	4	5	[1 2 3]	[1 2 3]
5	5	6	[1 2 3]	[1 2 3]
6	6	7	[1 2 3]	[1 2 3 6]

Rückgabewert der Funktion TEILER

Tabelle 3.2

Der letzte Wert von L wird als Funktionswert der Prozedur TEILER zurückgegeben. TEILER ist also im Gegensatz zu TEILER.AUSDRUCK eine Funktion und nicht nur eine Prozedur. Da die Variablen T und L als lokale Variable definiert waren, sind sie mit Ablauf der Funktion TEILER wieder gelöscht. Das einzige Ergebnis, welches dieser Lauf produziert hat, ist der Funktionswert der Funktion TEILER, den man auf verschiedene Weise weiterverarbeiten kann. Man kann zum Beispiel die Anzahl der Teiler bestimmen. Hierfür ist der Grundbefehl LAENGE nützlich, der die Elementezahl einer Liste zurückgibt.

LAENGE [A BC DEF]
ERGEBNIS: 3

LAENGE [1 2 3 4 5 [6 7 8 9 10]]
ERGEBNIS: 6

LAENGE [ ]
ERGEBNIS: 0

LAENGE TEILER 80
ERGEBNIS: 10

Bei dem letzten Aufruf ermittelte Logo zunächst den Wert von TEILER 80, also [1 2 4 5 8 10 16 20 40 80], und bestimmte dann die Länge dieser Liste.

Aufgabe 3.4: Schreibe eine Funktion TEILERZAHL.

Aufgabe 3.5: Experimentiere mit dem Aufruf LAENGE :W, wobei :W ein Logo-Wort ist; probiere LAENGE "DONAUDAMPFSCHIFFAHRTSKAPITAENSPATENT

Aufgabe 3.6: (Nur zur Demonstration)
Entferne die Zeile LOKAL "L in der Funktion TEILER und teste nach Ablauf der Prozedur den Befehl WERT "L (bzw. :L). Füge die Zeile nach dem Test wieder ein.

Hinweis: Es gibt eine noch elegantere Methode, Zeilen gelegentlich "kaltzustellen". Wenn irgendwo in einer Prozedurzeile ein **Semikolon** ("Strichpunkt") steht, wertet Logo den Rest der Zeile als **Kommentar**, der nicht auszuführen ist. Um eine Zeile kurzfristig auszuschalten, braucht man nur am Zeilenanfang ein Semikolon zu tippen, das sich später auch leicht wieder entfernen läßt.

Wie wir schon von der Prozedur TEILER.AUSDRUCK her wissen, können wir den Lauf der Funktion TEILER stark beschleunigen, indem wir mit jedem Teiler auch gleich seinen Partner abspeichern. Wir brauchen dann nur noch die Zahlen bis zur (Quadrat-) Wurzel von :N zu testen. Bei :N = 1000 brauchen wir nur 31 (anstatt 1000); bei :N = 1 000 000 nur 1000 (anstatt 1 000 000) Zahlen zu untersuchen. Je größer :N wird, desto mehr sparen wir ein!

```
PR TEILER.SCHNELLER  :N
 LOKAL  "L
 SETZE  "L  [ ]
 LOKAL  "T
 SETZE  "T  1
 SCHLEIFENANFANG:
 WENN  TEILT?  :T  :N  DANN  ________________________
 ___   ( SETZE  "L  MITLETZTEM  :T  :L )  ______________
 ___   ( SETZE  "L  MITLETZTEM  (DIV  :N  :T )  :L )
 SETZE  "T  :T + 1
 WENN :T * :T > :N  DANN  RUECKGABE :L
 GEHE  "SCHLEIFENANFANG
ENDE
```

Erläuterungen zur Prozedur TEILER.SCHNELLER: Zunächst sei daran erinnert, daß durch das Unterstreichungszeichen mehrere optische Zeilen zu einer logischen Zeile zusammengefaßt werden (siehe Prozedur: TEILT?.UMSTAENDLICH).

Die runden Klammern dienen in dieser Prozedur nur der optischen Zusammenfassung von Teilbefehlen. Im obigen Beispiel wären sie eigentlich entbehrlich. Logo würde die Prozedur ohne Klammern genau so ausführen wie mit den Klammern. Auch in Zukunkft wollen wir der besseren Lesbarkeit halber reichlich Klammern setzen. In Zweifelsfällen ist es immer ratsam, Klammern zu verwenden.

Einige Probeläufe unserer schnelleren Prozedur ergeben:

TEILER.SCHNELLER 92
ERGEBNIS: [1 92 2 46 4 23]

TEILER.SCHNELLER 36
ERGEBNIS: [1 36 2 18 3 12 6 6]

Es kann passieren, daß ein Teiler gleich seinem Partner ist.

Aufgabe 3.7: Welche Zahlen haben die Eigenschaft, daß einer ihrer Teiler gleich seinem Partner ist?

In solchen Fällen dürfen wir nur den Teiler, nicht aber den Partner abspeichern! Dies wollen wir eben noch korrigieren. Wir müssen also der bisherigen Teilerliste entweder den neuen Teiler mit seinem Partner oder nur den neuen Teiler hinzufügen. Damit die Teiler-Prozedur nicht zu sehr durch die notwendigen Fallunterscheidungen aufgebläht wird, schreiben wir am besten eine Hilfsfunktion, die uns genau die richtige Ergänzung liefert.

```
PR NEUE.TEILER  :T  :N
 WENN :T * :T = :N  DANN RUECKGABE   ( LISTE :T )
 RUECKGABE  ( LISTE  :T  ( DIV  :N  :T ) )
ENDE
```

Erläuterungen zur Prozedur NEUE.TEILER: Der Logo-Grundbefehl LISTE akzeptiert eine unterschiedliche Anzahl von Eingabeobjekten, die er alle zu einer Liste zusammenfügt (siehe Kapitel 2, Abschnitt 2.7). Hier noch einige Beispiele aus der Teilbarkeitslehre im Direktbetrieb:

SETZE "A 2
SETZE "B 12

(LISTE :A)
ERGEBNIS: [2]

(LISTE :A (DIV :B :A))
ERGEBNIS: [2 6]

Aufgabe 3.8: Begründe, daß die Zahlen :T und DIV :N :T genau dann zusammenfallen, wenn :T * :T = :N ist.

Hier nun die verbesserte Version der schnelleren TEILER-Prozedur:

```
 1: PR TEILER.SCHNELLER.VERBESSERT  :N
 2:  LOKAL  "L  SETZE  "L  [ ]
 3:  LOKAL  "T  SETZE  "T  1
 4:  SCHLEIFENANFANG:
 5:  WENN  TEILT?  :T  :N  DANN
 6:       SETZE  "L  ( SATZ  :L  NEUE.TEILER  :T  :N )
 7:  SETZE  "T  :T + 1
 8:  WENN  :T * :T > :N  DANN  RUECKGABE :L
 9:  GEHE  "SCHLEIFENANFANG
10: ENDE
```

Zunächst überprüfen wir, ob es tatsächlich klappt:

TEILER.SCHNELLER.VERBESSERT 81
ERGEBNIS: [1 81 3 27 9]

TEILER.SCHNELLER.VERBESSERT 1
ERGEBNIS: [1]

TEILER.SCHNELLER.VERBESSERT 56
ERGEBNIS: [1 56 2 28 4 14 7 8]

Erläuterungen zur Prozedur TEILER.SCHNELLER.VERBESSERT: Die zweite und dritte Zeile zeigen, daß man auch mehrere Anweisungen in eine Zeile schreiben

kann. Der Logo-Grundbefehl SATZ (siehe Kapitel 2, Abschnitt 2.7) fügt zwei Listen zu einer Gesamtliste zusammen. Durch die Anweisung (Zeile 6):

SETZE "L (SATZ :L NEUE.TEILER :T :N)

wird der alte Inhalt der Liste L mit dem Ergebnis des Aufrufs NEUE.TEILER :T :N "verschmolzen".

Aufgabe 3.9: Schreibe eine "Kosmetik-"Prozedur des Namens TEILER.TABELLE.IM.SCHOENDRUCK, mit der die bisherigen Teiler-Listen im "Schöndruck" ausgedruckt werden können. Der Aufruf TEILER.TABELLE.IM.SCHOENDRUCK 2310 soll etwa den folgenden Ausdruck zur Folge haben:

```
DIE TEILER DER ZAHL:  2310

        TEILER          PARTNER
------------------------------
            1            2310
            2            1155
            3             770
            5             462
            6             385
            7             330
           10             231
           11             210
           14             165
           15             154
           21             110
           22             105
           30              77
           33              70
           35              66
           42              55
```

Tabelle 3.3

Aufgabe 3.10: Schreibe eine Funktion TEILERLISTE, die zwar die "Partner-Methode" verwendet, die Teiler aber in einer der Größe nach geordneten Liste ausgibt; zum Beispiel:

TEILERLISTE 56

ERGEBNIS: [1 2 4 7 8 14 28 56]

(Hinweis: Das geht in diesem Fall auf direktem Weg - ohne zu sortieren.)

Aufgabe 3.11: Schreibe alle bisherigen Prozeduren dieses Kapitels, in denen der GEHE-Befehl vorkommt in endrekursive Prozeduren um. Hinweis: die endrekursiven Versionen sollten sich zu den "Sprung"-Versionen etwa so verhalten, wie die Prozedur SCHACHTABELLE.ER.V2 zur Prozedur SCHACHTABELLE.SPRUNG in Kapitel 2.

Wenn man keinen Computer hat, dann kann man Teilbarkeitstests auch mit Hilfe *spezieller Teilbarkeitsregeln* durchführen. Im folgenden sind einige dieser Regeln wiedergegeben. Sie gelten natürlich für die übliche Schreibweise im Zehnersystem. (Beachte dabei, daß die Zahl 0 durch jede Zahl teilbar ist).

Zweier-Regel: Eine Zahl ist durch 2 teilbar, wenn ihre letzte Ziffer durch 2 teilbar ist.

Dreier-Regel: Eine Zahl ist durch 3 teilbar, wenn ihre Quersumme durch 3 teilbar ist.

Die **Quersumme** einer Zahl ist einfach die Summe ihrer Ziffern. Die Quersumme der Zahl 72325 ist zum Beispiel 19.

Fünfer-Regel: Eine Zahl ist durch 5 teilbar, wenn ihre letzte Ziffer durch 5 teilbar ist.

Neuner-Regel: Eine Zahl ist durch 9 teilbar, wenn ihre Quersumme durch 9 teilbar ist.

Elfer-Regel: Eine Zahl ist durch 11 teilbar, wenn ihre alternierende Quersumme durch 11 teilbar ist.

Die **alternierende Quersumme** einer Zahl erhält man, wenn man die Ziffern der Zahl von rechts nach links abwechselnd mit negativem und positivem Vorzeichen aufsummiert. Alle Ziffern an den ungeraden Stellen (von rechts gezählt) werden subtrahiert, die an der geraden Stellen werden addiert. Dies wird am besten durch ein Beispiel deutlich. Die alternierende Quersumme der Zahl 82837293 wird so berechnet:

$$(-3) + 9 + (-2) + 7 + (-3) + 8 + (-2) + 8 = 22$$

Da die Zahl 22 durch 11 teilbar ist, folgt aus der Elfer-Regel, daß auch 82837293 durch 11 teilbar ist. Tatsächlich ist 82837293 = 11 * 7530663.

Wir wollen im folgenden einmal die Funktion QUERSUMME als Logo-Funktion schreiben. Da die eingegebenen Zahlen kaum mehr als 50 Ziffern haben dürften, können wir uns hier einmal den Luxus einer voll rekursiven Funktion leisten, ohne daß sie andauernd den Speicher zum Überlaufen bringt.

```
PR QUERSUMME  :N
 WENN LEER? :N  DANN RUECKGABE 0
 RUECKGABE  ( ERSTES :N ) + QUERSUMME OHNEERSTES :N
ENDE
```

Erläuterungen zur Prozedur QUERSUMME: Die Logo-Grundfunktionen LEER?, ERSTES und OHNEERSTES sind sowohl auf Wörter als auch auf Listen anwendbar (siehe Kapitel 1, Abschnitt 1.7 und Kapitel 2, Abschnitt 2.7). Da Logo auch Zahlen als Wörter auffasst, können diese Funktionen auch auf Zahlen angewandt werden. Hierzu noch einige Beispiele im Direktbetrieb:

ERSTES 739
ERGEBNIS: 7

OHNEERSTES 4711
ERGEBNIS: 711

Beachte die folgenden Beispiele genau:

ERSTES 1999999999 (9 Neuner)
ERGEBNIS: 1

ERSTES 19999999999 (10 Neuner)
ERGEBNIS: 2

Was ist hier passiert? Erinnerst du dich an das, was in Kapitel 1 zu den in Logo verfügbaren Zahlen gesagt wurde? Dann weißt du auch, daß die Zahl 19999999999 den Bereich der in Logo zugelassenen ganzen Zahlen überschreitet. Diese Zahl wird unmittelbar nach ihrer Eingabe in das Gleitkommaformat umgesetzt. Aber dieses Format verfügt nur über sechs gültige Ziffern. Also wird die eingegebene Zahl durch die am nächsten liegende Gleitkommazahl ersetzt. Dies ist die Zahl 2E10. Die Funktion ERSTES, angewandt auf 2E10, liefert nun das Ergebnis 2. Hübsch häßlich, nicht wahr?

Es gibt aber teilweise eine Rettung in Logo. Man kann Zahlen auch "zwangsweise" als Wörter (Zeichenketten) eingeben. Wie bei Wörtern üblich, muß man ihnen dann ein Anführungszeichen voranstellen. Wenn wir das tun, dann liefert die Funktion ERSTES immer das richtige Ergebnis.

ERSTES "8273
ERGEBNIS: 8

ERSTES "19999999999 (10 Neuner)
ERGEBNIS: 1

Man kann sogar mit "gequoteten" Zahlen rechnen:

"2 + "3
ERGEBNIS: 5

Die Leerstellen vor und nach dem Additionszeichen sind in diesem Beispiel sehr wichtig! Probiere:

"2+"3
ERGEBNIS: 2+"3

Hier wurde die Eingabe von Logo nicht als ein Rechenausdruck erkannt, sondern als das vierstellige Wort 2+"3. Und wenn im Direktbetrieb ein Wort eingegeben wird, dann gibt Logo das Wort unverändert als Ergebnis zurück; zum Beispiel:

"ABCD
ERGEBNIS: ABCD

Auch folgendes ist noch möglich:

QW "25
ERGEBNIS: 5

Es wäre aber ein Irrtum anzunehmen, daß man mit dieser "Wort-Arithmetik" von Logo die Beschränkungen im ganzzahligen Bereiche überwunden hätte. Wenn man Zahlen, die den Ganzzahlbereich überschreiten, als Worte eingibt und mit ihnen rechnet, dann überführt Logo diese Zahlen trotz der Anführungszeichen in Gleitkommazahlen. Ein Beispiel:

"123456789101112 + "121110987654321
ERGEBNIS: 2.44567E14

Zurück zur Funktion QUERSUMME. Zum Verständnis derartiger voll rekursiver Funktionen kann der Protokoll-Modus sehr hilfreich sein. Um dies besser verfolgen zu können, wollen wir die Kurzversionen der Logo-Grundwörter verwenden; also

ER an Stelle von ERSTES, OE an Stelle von OHNEERSTES und RG an Stelle von RUECKGABE. Auf das DANN verzichten wir ganz.

Aufgabe 3.12: Verändere die Prozedur QUERSUMME im Editor so, daß die Kurzformen der Logo-Grundwörter verwendet werden und streiche das Grundwort DANN ganz.

Probiere nun folgendes und ähnliche Beispiele aus:

```
PE
QUERSUMME  "8357
AUFRUF--- QUERSUMME  8357
  WENN LEER? :N RG 0
  RG ( ER :N ) + QUERSUMME OE :N
 AUFRUF--- QUERSUMME 357
   WENN LEER? :N RG 0
   RG ( ER :N ) + QUERSUMME OE :N
  AUFRUF--- QUERSUMME 57
    WENN LEER? :N RG 0
    RG ( ER :N ) + QUERSUMME OE :N
   AUFRUF--- QUERSUMME 7
     WENN LEER? :N RG 0
     RG ( ER :N ) + QUERSUMME OE :N
    AUFRUF--- QUERSUMME       (Eingabe: <leeres Wort>)
      WENN LEER? :N RG 0
     RUECKG.: 0
    FERTIG QUERSUMME
    RUECKG.: 7
   FERTIG QUERSUMME
   RUECKG.: 12
  FERTIG QUERSUMME
  RUECKG.: 15
 FERTIG QUERSUMME
 RUECKG.: 23
FERTIG QUERSUMME
ERGEBNIS: 23
```

In mathematischer Darstellungsform könnte man diesen Protokoll-Lauf folgendermaßen lesen:

```
QUERSUMME  8357
  =  8 + ( QUERSUMME 357 )
  =  8 + ( 3 + ( QUERSUMME 57 ) )
  =  8 + ( 3 + ( 5 + ( QUERSUMME 7 ) ) )
  =  8 + ( 3 + ( 5 + ( 7 + ( QUERSUMME <leeres Wort> ))))
  =  8 + ( 3 + ( 5 + ( 7 + 0 ) ) )
  =  8 + ( 3 + ( 5 + 7 ) )
  =  8 + ( 3 + 12 )
  =  8 + 15
  =  23
```

Die dritte Zeile in der Funktion QUERSUMME ist ein wichtiges Beispiel für die Verwendung von Klammern. Wenn wir diese Klammern weglassen, dann erhalten wir falsche Ergebnisse; zum Beispiel:

QUERSUMME 69
ERGEBNIS: 7

Der Unterschied ergibt sich daraus, daß die Addition stärker bindet als die Funktionsauswertung. Im Falle des Aufrufs ERSTES 253 + 698 wird zuerst die Summe von 253 und 698 (also 951) berechnet und von dieser Summe dann das erste Zeichen genommen. Das Ergebnis lautet also 9.

Das mathematische Protokoll sähe in diesem Fall folgendermaßen aus. Um deutlicher zu machen, was zuerst berechnet wird, setzten wir reichlich Klammern.

```
QUERSUMME 69  =  ER ( 69 + QUERSUMME 9 )
    =  ER ( 69 + ( ER ( 9 +  QUERSUMME <leeres Wort> ) ) )
    =  ER ( 69 + ( ER ( 9 + 0 ) ) )
    =  ER ( 69 + ( ER 9 ) )
    =  ER ( 69 + 9 )
    =  ER ( 78 )
    =  7
```

Aufgabe 3.13: Editiere die Funktion QUERSUMME und entferne die Klammern. Führe danach Tests wie den eben gezeigten durch. Korrigiere QUERSUMME hinterher wieder.

Die Quersumme einer Zahl kann manchmal auch noch ziemlich groß sein.

QUERSUMME "9876543210012345678998765432100123456789987654321
ERGEBNIS: 225

Wir können diese Ergebnis natürlich noch einmal in die Quersummen-Funktion einspeisen und erhalten dann die Quersumme der Quersumme:

QUERSUMME
QUERSUMME "9876543210012345678998765432100123456789987654321
ERGEBNIS: 9

Aufgabe 3.14: Schreibe je eine endrekursive und eine "Sprung"-Version der Quersummen-Funktion.

Aufgabe 3.15: Schreibe eine Funktion ITERIERTE.QUERSUMME, welche die Quersummenbildung so lange wiederholt, bis eine einstellige Zahl herauskommt.

Aufgabe 3.16: Schreibe eine Funktion ALTERNIERENDE.QUERSUMME.

Aufgabe 3.17: Schreibe eine Funktion ITERIERTE.ALTERNIERENDE.QUERSUMME.

Aufgabe 3.18: (Ein vergleichsweise anspruchsvolles Projekt)
Schreibe Funktionen WORTADDITION, WORTMULTIPLIKATION, WORTSUBTRAKTION und WORTDIVISION, die es erlauben, beliebig lange als Wörter eingegebene (d.h. gequotete) ganze Zahlen exakt zu verarbeiten.

Aufgabe 3.19: Schreibe eine Funktion TEILERSUMME :N, die alle Teiler einer Zahl aufsummiert; zum Beispiel:
TEILERSUMME 12 = 1+2+3+4+6+12 = 28

Aufgabe 3.20: Die Teilersumme einer von 1 verschiedenen Zahl a ist natürlich immer größer als die Zahl a selbst. Denn a gehört ja zu seinen Teilern, und es gibt noch weitere Teiler.

(a) Suche Zahlen a, für die gilt: Teilersumme von a < 2 * a
(b) Suche Zahlen b, für die gilt: Teilersumme von b > 2 * b
(c) Suche Zahlen c, für die gilt: Teilersumme von c = 2 * c

Die in (c) beschriebenen Zahlen heißen **vollkommene Zahlen** (perfekte Zahlen). Die Zahl 6 ist zum Beispiel eine vollkommene Zahl.

Aufgabe 3.21: Manche der in diesem Abschnitt geschriebenen Prozeduren liegen in verschiedenen Versionen vor. Führe einige systematische Tests zu ihrem Laufzeitverhalten durch. Entscheide dich, welche du davon als "Arbeitsversionen" behalten willst. Gib ihnen "griffige" Namen und speichere sie in einer Datei unter dem Namen TEILBARKEIT ab. Du mußt dabei die folgenden Schritte ausführen:

(1) Speichere zunächst zur Sicherheit alles unter dem Namen HILFSDATEI ab; der entsprechende Logo-Befehl lautet:

```
BEWAHRE "HILFSDATEI
```

Der Dateiname, hier HILFSDATEI, muß vorn mit Anführungszeichen versehen (gequotet) werden.

(2) Lösche alle globalen Namen mit Hilfe des Befehls VERGISS NAMEN .

(3) Lösche alle Prozeduren, die nicht in die Bibliothek aufgenommen werden sollen, mit Hilfe des VERGISS Befehls. Hier ein Beispiel:

```
VERGISS TEILT?.UMSTAENDLICH
```

(Beachte, daß der Prozedurname nicht gequotet, also nicht mit einem Anführungszeichen versehen wird).

(4) Gib den Arbeits-Prozeduren, die in die Bibliothek kommen sollen, griffige, prägnante Namen. Du kannst dies tun, in dem du die Prozedur, deren Namen du abändern willst, einfach noch einmal editierst und im Editor mit dem neuen Namen versiehst. Beachte aber, daß dies "stimmig" sein muß. Auch in anderen Prozeduren, welche die nun veränderte Prozedur aufgerufen haben, muß der Prozedurname entsprechend geändert werden. Dies gilt insbesondere auch für rekursive Aufrufe. Verlasse den Editor jeweils mit der RUN/STOP-Taste, so daß die Prozedur unter dem neuen Namen gelernt wird. Die betreffende Prozedur kommt jetzt zweimal vor: einmal unter dem alten und einmal unter dem neuen Namen.

(5) Lösche die Prozedur unter dem alten Namen so, wie es in Schritt (3) beschrieben wurde.

(6) Teste die verbleibenden Arbeitsprozeduren nochmals aus, bis sie in Ordnung sind.

(7) Speichere die Arbeits-Prozeduren mit Hilfe des Befehls

```
BEWAHRE "TEILBARKEIT
```

auf Diskette.

(8) Deine Arbeit ist zu wichtig, als daß du das Risiko eingehen solltest, sie durch eine Unachtsamkeit oder durch einen Zufall zu verlieren. Einzelne Dateien oder ganze Disketten können durch kleine Bedienungsfehler gelöscht oder überschrieben werden, Disketten können Fehler in der Magnetisierung aufweisen oder in streuende Magnetfelder geraten, sie können geknickt oder von Katzen angefressen werden. Speichere deshalb die Datei TEILBARKEIT nochmals auf einer zweiten Sicherheitsdiskette ab. Wenn eine der beiden Versionen zerstört werden sollte, dann stelle sofort wieder eine neue Sicherheitskopie her.

(9) Keine Panik, wenn du bei einem der obigen Schritte etwas falsch gemacht hast. Du sagst dann einfach ADE und danach

LADE "HILFSDATEI

und der Ausgangszustand vor Schritt (1) ist wieder hergestellt.

Nur keine Angst; das gesamte Verfahren ist weniger aufwendig, als es vielleicht auf den ersten Blick aussieht. Du wirst dich an derartige Vorgehensweisen schnell gewöhnen.

(10) Außer vielleicht zu Demonstrationszwecken sind die Prozeduren in der HILFSDATEI jetzt überflüssig. Wenn du glaubst, daß du sie nicht mehr brauchst, dann kannst du die HILFSDATEI mit Hilfe des Befehls

VERGISSDATEI "HILFSDATEI

löschen. Sie ist dann aber unwiederbringlich verloren.

3.2 Vielfache

Wer *Teiler* sagt, muß auch *Vielfache* sagen. Jede Zahl hat nur endlich viele Teiler, aber unendlich viele Vielfache. Wir können also nie alle Vielfachen einer Zahl abspeichern oder auch nur ausdrucken.

Die folgende Prozedur druckt die Vielfachen von :A aus, ohne aufzuhören. Man muß sie mit dem Kommando Control-G (CTRL-G) unterbrechen.

```
PR VIELFACHE.AUSDRUCK :A
 VIELFACHE.AUSDRUCK.HP 1  :A
ENDE

PR VIELFACHE.AUSDRUCK.HP  :I  :A
 ( DRUCKEZEILE  :I  :I*:A )
 VIELFACHE.AUSDRUCK.HP  (:I+1)  :A
ENDE
```

```
VIELFACHE.AUSDRUCK 17
1 17
2 34
3 51
4 68
 ...
```

Erläuterungen zur Prozedur VIELFACHE.AUSDRUCK: Sie dient nur dazu, um das eigentliche "Arbeitspferd", die Prozedur VIELFACHE.AUSDRUCK.HP, mit den richtigen Anfangswerten aufzurufen. Die Hilfsvariable I stellt einen Zähler dar, der uns angibt, mit welchem Vielfachen wir es gerade zu tun haben, :A ist die Ausgangszahl, deren Vielfache ausgedruckt werden. Bei jedem rekursiven Aufruf wird der Wert von I um 1 erhöht; der Wert von A bleibt natürlich unverändert. (Der Parameter :A darf beim rekursiven Aufruf aber nicht einfach weggelassen werden, da sonst die Anzahl der Eingabeparameter nicht stimmen würde).

Aufgabe 3.22: Schreibe eine "Sprung"-Version der obigen Prozedur.

Wenn man die Vielfachen weiterverarbeiten will, dann muß man sie abspeichern; zum Beispiel in einer Liste. Es wird jetzt unumgänglich, daß wir einen Wert :OBERGRENZE eingeben, bis zu dem die Vielfachen gesammelt werden.

```
 1:  PR VIELFACHE  :A  :OBERGRENZE
 2:   LOKAL  "VL
 3:   SETZE  "VL  ( LISTE  :A )
 4:   LOKAL  "V
 5:   SETZE  "V  :A
 6:   MARKIERUNG1:
 7:   SETZE  "V  :V + :A
 8:   WENN  :V > :OBERGRENZE  DANN RUECKGABE  :VL
 9:   SETZE  "VL  ( MITLETZTEM  :V  :VL )
10:   GEHE  "MARKIERUNG1
11:  ENDE
```

VIELFACHE 19 200
ERGEBNIS: [19 38 57 76 95 114 133 152 171 190]

VIELFACHE 23 276
ERGEBNIS: [23 46 69 92 115 138 161 184 207 230 253 276]

Erläuterungen zur Prozedur VIELFACHE: Die Zeilennummern gehören nicht zur Funktion, sondern dienen nur dazu, die einzelnen Zeilen für die Erläuterungen besser "dingfest" zu machen. Zunächst wird der Name VL für eine Hilfsliste eingeführt. Die Liste :VL wird in Zeile 3 zunächst mit der Zahl :A bestückt. Die Variable V steht für Vielfache. In Zeile 5 erhält sie den Wert :A zugewiesen; in Zeile 7 wird ihr augenblicklicher Wert jeweils um :A erhöht. Dies bedeutet, daß wir zum nächsten Vielfachen von :A übergehen. In Zeile 9 wird der Variablen VL der Wert des Aufrufs MITLETZTEM :V :VL zugewiesen. Dabei wird der aktuelle Wert von V als letztes Element in die bisherige Liste VL eingefügt. Die Klammern sind eigentlich überflüssig; sie dienen nur der optischen Gruppierung der Befehlsfolge.

Aufgabe 3.23: Schreibe eine endrekursive Version der Funktion VIELFACHE.

Aufgabe 3.24: Wir wollen jetzt unsere Datei TEILBARKEIT aus dem vorigen Abschnitt durch die neuen Prozeduren erweitern. Du kannst dich dabei stark an die Vorgehensweise in Aufgabe 3.21 anlehnen. Hier nur die wichtigsten Schritte.

(1) Speichere zunächst sicherheitshalber alles ab; etwa durch den Befehl

BEWAHRE "HILFSDATEI

(2) Bereinige den Zustand der Arbeitsumgebung (die *Arbeitsumgebung* ist alles das, was Logo auf den Befehl ZEIGE ALLES ausdruckt) folgendermaßen:

* Lösche, falls vorhanden, alle globalen Namen.
* Lösche alle überflüssigen Prozeduren.
* Gib den verbleibenden Prozeduren griffige Namen. (Achte darauf, daß keiner der neuen Prozedurnamen mit einem Namen der bisherigen Teilbarkeits-Datei zusammenfällt).

(3) Lade die alte Teilbarkeits-Datei mit dem Kommando

LADE "TEILBARKEIT

(Falls alte Prozedurnamen aus dieser Datei mit den Prozedurnamen der neuen im Arbeitsspeicher befindlichen Prozeduren zusammenfallen würden, würden die neuen Prozeduren beim Laden durch die alten Prozeduren überschrieben und wären verloren).

(4) Schaue dir sicherheitshalber die erweiterte Teilbarkeitsumgebung mit Hilfe des Befehls

ZEIGE ALLES

an. Korrigiere den Stand, falls notwendig.

(5) Speichere die neue Teilbarkeitsumgebung unter dem Namen TEILBARKEIT ab. Die alte Datei desselben Namens wird dabei überschrieben. (Du kannst das Überschreiben der alten Version vermeiden, indem du der neuen Version einen anderen Namen gibst).

(6) Lege eine Sicherheitskopie der Teilbarkeits-Bibliothek an.

(7) Lösche unter Umständen die zunächst aus Sicherheitsgründen angelegte HILFSDATEI.

3.3 Primzahlen

Manche Zahlen haben ziemlich wenige Teiler. Wir finden einige davon mit Hilfe der TEILER-Funktion aus Abschnitt 3.1.

TEILER 2
ERGEBNIS: [1 2]

TEILER 3
ERGEBNIS: [1 3]

TEILER 5
ERGEBNIS: [1 5]

TEILER 17
ERGEBNIS: [1 17]

TEILER 59
ERGEBNIS: [1 59]

TEILER 509
ERGEBNIS: [1 509]

Jede natürliche Zahl (außer 1) hat mindestens zwei Teiler, nämlich die Zahl 1 und sich selbst. Zahlen, welche nur diese beiden "trivialen" Teiler besitzen, heißen **Primzahlen.** (Die Zahl 1 wird nicht zu den Primzahlen gerechnet, obwohl sie auch nur triviale Teiler hat). Primzahlen sind die Bausteine aller Zahlen. Jede von 1 verschiedene natürliche Zahl ist nämlich in ein Produkt von Primzahlen zerlegbar; zum Beispiel 45 = 3 * 3 * 5. Nur die Primzahlen selbst sind nicht weiter in Faktoren zerlegbar.

Die Grund-Funktion LAENGE und die selbstgeschriebene Funktion TEILER ermöglichen den folgenden Primzahltest:

```
PR PRIMZAHL?  :A
 RUECKGABE  ( LAENGE  TEILER :A ) = 2
ENDE
```

PRIMZAHL? 7
ERGEBNIS: WAHR

PRIMZAHL? 15
ERGEBNIS: FALSCH

Aufgabe 3.25: Schreibe die Funktion PRIMZAHL? so um, daß sie auf die Funktion TEILERZAHL zurückgreift (siehe Aufgabe 3.4).

Aufgabe 3.26: Die Prüf-Funktion PRIMZAHL? tut zwar, was von ihr erwartet wird; aber es ist doch ziemlich aufwendig, zuerst die gesamte Teilermenge einer Zahl zu bestimmen, um entscheiden zu können, ob es sich um eine Primzahl handelt. Schreibe eine Prüffunktion PRIM?, mit welcher auf direktem Wege getestet werden kann, ob der Eingabewert eine Primzahl ist. (Hinweis: Überprüfe die Teilbarkeit durch 2 gesondert; starte mit dem Anfangswert 3 für die Testvariable; gehe in Zweierschritten vorwärts; beende den Test spätestens bei der Quadratwurzel von :N).

Beispiel: Die Zahl 1034971 ist durch 7 teilbar und somit keine Primzahl. Mit dem soeben beschriebenen Test findet man das sehr schnell heraus. Denn 7 ist bereits die vierte Testzahl nach 2, 3 und 5. Die Ermittlung aller Teiler und die Bestimmung der Teilerzahl dauern dagegen erheblich länger.

Aufgabe 3.27: Schreibe eine Funktion KLEINSTER.PRIMTEILER :N, die als Funktionswert den kleinsten Primteiler der Zahl :N liefert. (Der kleinste Primteiler ist natürlich der kleinste Teiler, der zugleich eine Primzahl ist). Beachte, daß die Zahl 0 überhaupt keinen Primteiler besitzt und daß der kleinste Primteiler einer Primzahl die Primzahl selbst ist.

Mit Hilfe dieser Funktion können wir leicht eine Funktion zur Primfaktorzerlegung einer Zahl schreiben:

```
PR PRIMFAKTORZERLEGUNG  :N
 WENN  :N < 2  DRUCKEZEILE [ EXISTIERT NICHT ]  AUSSTIEG
 LOKAL "L
 SETZE  "L  [ ]
 LOKAL "T
 SCHLEIFENBEGINN:
 WENN  :N = 1  DANN  RUECKGABE :L
 SETZE  "T  KLEINSTER.PRIMTEILER :N
 SETZE  "L  MITLETZTEM  :T  :L
 SETZE  "N  DIV  :N  :T
 GEHE  "SCHLEIFENBEGINN
ENDE
```

Ein Beispiel:

PRIMFAKTORZERLEGUNG 300300
ERGEBNIS: [2 2 3 5 5 7 11 13]

Es ist also 300300 = 2 * 2 * 3 * 5 * 5 * 7 * 11 * 13.

Erläuterung der Prozedur PRIMFAKTORZERLEGUNG: Zunächst zur Erinnerung: Ist der Eingabewert von DRUCKE bzw. DRUCKEZEILE eine Liste, so druckt Logo die einzelnen Listenelemente, läßt aber die äußersten Listenklammern weg (siehe Kapitel 1, Abschnitt 1.8). Dies erleichtert das Schreiben von Dialogprogrammen. Das Grundwort AUSSTIEG bewirkt, daß Logo seine gesamte Arbeit abbricht und in den Direktbetrieb zurückkehrt. Die Verwendung von AUSSTIEG ist meist unvermeidlich, wenn irgendein irreparabler Fehler angetroffen wurde. Im obigen Fall wurde es als Fehler angesehen, daß die Primfaktorzerlegung von einer (natürlichen) Zahl verlangt wurde, die kleiner als 2 ist. Die Zeile

```
WENN :N < 2  DRUCKEZEILE [ EXISTIERT NICHT ]  AUSSTIEG
```

ist wieder ein Beispiel dafür, daß man das Grundwort DANN eigentlich nicht benötigt. Es dient nur der besseren Lesbarkeit. Solche Grundwörter bezeichnet man auch als "syntaktischen Zucker". Man hätte auf diese Zeile auch ganz verzichten können. Allerdings hätte dann an Stelle von

```
WENN  :N = 1  DANN  RUECKGABE :L
```

die folgende Bedingung formuliert werden müssen:

```
WENN  :N < 2  DANN  RUECKGABE :L
```

Der Aufruf PRIMFAKTORZERLEGUNG 0 bzw. PRIMFAKTORZERLEGUNG 1 hätte dann als Ergebnis jeweils die leere Liste gehabt.

Aufgabe 3.28: Ändere die Prozedur PRIMFAKTORZERLEGUNG so ab, daß für :N = 0 und :N = 1 keine Fehlermeldung ausgedruckt, sondern die leere Liste als Ergebnis zurückgegeben wird.

Zum Experimentieren ist es manchmal günstig, wenn man für eine Prozedur, die einen langen Namen hat, eine Version mit Kurznamen schreibt. Dazu sollte man aber nicht noch einmal die gesamte Prozedur mit dem langen Namen kopieren, sondern nur einen Aufruf für die Prozedur mit dem langen Namen schreiben. Eine Kurzform der Prozedur PRIMFAKTORZERLEGUNG könnte so aussehen:

```
PR PFZ  :N
 RUECKGABE  PRIMFAKTORZERLEGUNG  :N
ENDE
```

Aufgabe 3.29: Schreibe entsprechende Kurzformen für die anderen Prozeduren dieses Kapitels; zum Beispiel: KPT an Stelle von KLEINSTER.PRIMTEILER, usw.

Die Primzahlen sind einer der ältesten und interessantesten Untersuchungsgegenstände der Mathematik. Schon im Altertum war man bestrebt, einen möglichst guten Überblick über die Primzahlen zu gewinnen. Der griechische Mathematiker *Eratosthenes von Kyrene* (etwa 276 - 195 v.Chr.) gab das folgende Verfahren an, um alle Primzahlen bis zu einer bestimmten vorgegebenen Zahl n zu bestimmen. Es sei hier am Beispiel n = 20 erläutert.

(1) Schreibe alle Zahlen von 1 bis 20 auf:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

(2) Streiche die Zahl 1.

~~1~~ 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

(3) Unterstreiche die Zahl 2.

~~1~~ $\underline{2}$ 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

(4) Streiche alle echten Vielfachen von 2; also die Zahlen 4, 6, 8, 10, 12, 14, 16, 18 und 20.

~~1~~ $\underline{2}$ 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~

(5) Unterstreiche die erste freie (d.h. noch nicht unterstrichene oder gestrichene) Zahl; in diesem Fall also die Zahl 3.

~~1~~ $\underline{2}$ $\underline{3}$ ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~

(6) Streiche alle echten Vielfachen von 3; also die Zahlen 6, 9, 12, 15 und 18. Wenn dabei eine schon durchgestrichene Zahl nochmals durchgestrichen wird (wie zum Beispiel die Zahl 6), so macht das nichts.

~~1~~ $\underline{2}$ $\underline{3}$ ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~

(7) Unterstreiche die nächste freie Zahl; in diesem Fall also die Zahl 5.

~~1~~ $\underline{2}$ $\underline{3}$ ~~4~~ $\underline{5}$ ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~

(8) Streiche alle echten Vielfachen der Zahl 5; also die Zahlen 10, 15 und 20.

~~1~~ $\underline{2}$ $\underline{3}$ ~~4~~ $\underline{5}$ ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~

(9) Setze das Verfahren so lange fort, bis jede der Zahlen entweder unterstrichen oder durchgestrichen ist.

~~1~~ $\underline{2}$ $\underline{3}$ ~~4~~ $\underline{5}$ ~~6~~ $\underline{7}$ ~~8~~ ~~9~~ ~~10~~ $\underline{11}$ ~~12~~ $\underline{13}$ ~~14~~ ~~15~~ ~~16~~ $\underline{17}$ ~~18~~ $\underline{19}$ ~~20~~

(10) Ende des Verfahrens. Die unterstrichenen Zahlen sind die Primzahlen zwischen 1 und 20:

2 3 5 7 11 13 17 19

Durch dieses Verfahren werden also genau die Primzahlen "ausgesiebt". Man nennt das Verfahren deshalb auch das **Sieb des Eratosthenes**.

Aufgabe 3.30: Gib eine allgemeine Beschreibung des Siebverfahrens, die von der Zahl 20 unabhängig ist.

Aufgabe 3.31: Schritt Nr. (9) im obigen Verfahren ist etwas umständlich. Im Grunde genommen sind wir schon nach Schritt Nr. (8) fertig gewesen. Schreibe das Siebverfahren so auf, daß dies berücksichtigt wird. Der Algorithmus wird dadurch schneller.

Wir wollen das Siebverfahren jetzt in ein Logo-Programm umsetzen. Dabei gibt es ein kleines Problem. Beim oben beschriebenen Siebverfahren kann man sich vorstellen, daß man mit Papier und Bleistift arbeitet oder daß man gar die Zahlen mit einem Stöckchen in den Sand schreibt. Der Sand oder das Papier speichern die aufgeschriebenen Zahlen. Beim Arbeiten mit dem Computer müssen wir sie auf eine andere Weise speichern. Hierfür bieten sich wieder die Listen an. Die Ausgangsliste könnte im obigen Beispiel so aussehen:

[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]

Auch das Unterstreichen und das Durchstreichen müssen wir in Logo auf andere Art umsetzen. Es gibt verschiedene Möglichkeiten, dies zu tun. In unserer ERATOSTHENES-Funktion wird es so gemacht, daß die echten Vielfachen der Primzahlen nicht durchgestrichen, sondern statt dessen "auf Null gesetzt" werden, und daß die unterstrichenen Zahlen einfach stehen bleiben. Zum Schluß sieht die "gesiebte" Liste also so aus:

[0 2 3 0 5 0 7 0 0 0 11 0 13 0 0 0 17 0 19 0]

Die in der Liste enthaltenen von Null verschiedenen Zahlen sind also gerade die gesuchten Primzahlen.

Bevor wir mit dem Aussieben beginnen können, benötigen wir eine Funktion, die uns die Ausgangsliste liefert. Da in der Ausgangsliste die "Primzahlkandidaten" stehen, wollen wir die Funktion KANDIDATENLISTE nennen.

```
PR KANDIDATENLISTE  :N
 LOKAL  "KL
 SETZE  "KL  [ ]
 LOKAL  "I
 SETZE  "I  1
 SCHLEIFENBEGINN:
 WENN  :I > :N  DANN RUECKGABE  :KL
 SETZE  "KL  MITLETZTEM  :I  :KL
 SETZE  "I  :I +1
 GEHE  "SCHLEIFENBEGINN
ENDE
```

Einige Beispiel im Direktbetrieb:

KANDIDATENLISTE 6
ERGEBNIS: [1 2 3 4 5 6]

KANDIDATENLISTE 20
ERGEBNIS: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]

Aufgabe 3.32: Schreibe eine endrekursive Version der Funktion KANDIDATEN-LISTE.

Nun zum Siebverfahren selbst. Die entsprechende Funktion soll ERATOSTHENES heißen.

```
 1: PR ERATOSTHENES :N
 2:  LOKAL  "KL
 3:  SETZE "KL  UEBERSCHREIBE 1  ( KANDIDATENLISTE :N )  0
 4:  LOKAL  "I
 5:  SETZE  "I  1
 6:  LOKAL  "V
 7:  MARKIERUNG1:
 8:  ( DRUCKEZEILE  "STUFE  :I  ": )   DRUCKEZEILE  :KL
 9:  WENN  :I * :I  >  :N  DANN  RUECKGABE  :KL
10:  SETZE  "I  :I + 1
11:  WENN ( ELEMENT :I :KL ) = 0 DANN GEHE "MARKIERUNG1
12:  SETZE  "V  :I + :I
13:  MARKIERUNG2:
14:  WENN  :V > :N  DANN GEHE "MARKIERUNG1
15:  SETZE  "KL  UEBERSCHREIBE  :V  :KL  0
16:  SETZE  "V  :V + :I
17:  GEHE  "MARKIERUNG2
18: ENDE
```

Erläuterungen zur Prozedur ERATOSTHENES: Zunächst werden in ERATOSTENES einige (lokale) Hilfsvariable definiert. KL ist eine Hilfsvariable für die Kandidatenliste, I für die der Reihe nach zu durchlaufenden Zahlen und V für die Vielfachen von I. UEBERSCHREIBE ist eine weiter unten gegebene Hilfsfunktion mit drei Eingaben: einer Zahl :N, einer Liste :L und einem Objekt :X. UEBERSCHREIBE ersetzt das Element Nr. :N der Liste :L durch das Objekt :X. Der Befehl

UEBERSCHREIBE 1 (KANDIDATENLISTE :N) 0

bewirkt also, daß das erste Element (also die 1) in der Kandidatenliste "ausgenullt" wird, denn 1 ist keine Primzahl. In Zeile 15 bewirkt der Befehl UEBERSCHREIBE :V :KL 0, daß das gegenwärtige Vielfache :V der Zahl :I ausgenullt wird.

In Zeile Nr. 8 werden die bisher erreichten Zwischenergebnisse ausgedruckt. Diese Zeile ist für das Gesamtergebnis der Funktion ERATOSTHENES im Grunde genommen überflüssig. Aber der Verlauf des Siebverfahrens wird durch den Ausdruck der Zwischenergebnisse sehr gut im einzelnen erkennbar. Besonders beim Testen einer Prozedur kann das Ausdrucken von Zwischenergebnissen sehr bei der Fehlersuche helfen. Es stellt insbesodere auch eine Alternative zum Protokollmodus dar, von dessen Informationsfülle der Benutzer manchmal "erschlagen" wird.

Der Befehl (DRUCKEZEILE "STUFE :I ":) ist wieder ein Beispiel dafür, daß man mit einem Druckbefehl mehrere Daten ausdrucken kann; in diesem Fall das

Wort STUFE, den Wert der Zahl I und den Doppelpunkt. Die Leerstelle zwischen dem Doppelpunkt und der schließenden Klammer ist sehr wichtig.

Im Eingangsbeispiel (:N = 20) hätten wir eigentlich bei der Zahl 5 aufhören können, denn ab dieser Zahl wurde keine zu streichende Zahl mehr angetroffen, die nicht vorher schon gestrichen worden war. Das Verfahren mußte nur deshalb noch weitergeführt werden, damit zum Schluß alle Primzahlen unterstrichen waren. Die Kennzeichnung der Nichtprimzahlen durch das Ausnullen verschafft uns hier Vorteile. Denn die Primzahlen sind diejenigen Zahlen, die zum Schluß von Null verschieden sind. Die hat man aber auch schon beim Durchlauf der Zahl 5 (bei :N = 20). Im allgemeinen Fall braucht man das Ausnullen der Vielfachen von :I nur "bis zur Wurzel von :N", also bis zu dem kleinsten :I mit :I * :I > :N durchzuführen. Diese Stop-Bedingung ist in Zeile 9 zu finden; zum Schluß wird als Funktionswert die bearbeitete Kandidatenliste zurückgegeben.

Zeile 10 stellt den jeweiligen Übergang zur nächsten zu untersuchenden Zahl dar. Durch Zeile 11 wird verhindert, daß Vielfache von schon gestrichenen Zahlen nochmals gestrichen werden. Es wäre ja zum Beispiel sinnlos, die Vielfachen von 4 zu streichen, da diese schon als Vielfache von 2 gestrichen worden sind. Das Logo-Grundwort ELEMENT ermöglicht einen lesenden Direktzugriff auf die Elemente einer Liste. Ein Beispiel im Direktbetrieb:

```
ELEMENT 4 [ ANNA EMMA HUGO ISOLDE OTTO XAVER ]
ERGEBNIS: ISOLDE
```

In Zeile 12 wird die Ausnullung der Vielfachen von :I vorbereitet. Da die Zahl :I selbst nicht ausgenullt werden soll, wird beim Doppelten von :I begonnen. Die Ausnullung dieser Vielfachen spielt sich in der durch MARKIERUNG2: gekennzeichneten Schleife ab (Zeile 13 bis Zeile 17). Zeile 14 sorgt für den richtigen Ausstieg aus dieser Schleife. In Zeile 16 wird zum jeweils nächsten Vielfachen von :I übergegangen.

In der Funktion ERATOSTHENES tritt die Schleifenbildung in verschachtelter Form auf. Die Schleife von Zeile 13 bis Zeile 17 ist komplett in der Schleife von Zeile 7 bis Zeile 17 enthalten.

Ein Lauf der Funktion ERATOSTHENES sieht zum Beispiel so aus:

```
ERATOSTHENES  20
STUFE 1:
0 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
STUFE 2:
0 2 3 0 5 0 7 0 9  0 11  0 13  0 15  0 17  0 19  0
STUFE 3:
0 2 3 0 5 0 7 0 0  0 11  0 13  0  0  0 17  0 19  0
STUFE 4:
0 2 3 0 5 0 7 0 0  0 11  0 13  0  0  0 17  0 19  0
STUFE 5:
0 2 3 0 5 0 7 0 0  0 11  0 13  0  0  0 17  0 19  0
ERGEBNIS:
[ 0 2 3 0 5 0 7 0 0  0 11  0 13  0  0  0 17  0 19  0 ]
```

Der Probelauf macht einige Dinge besonders deutlich: auf STUFE 3 folgt sehr schnell STUFE 4. Wir erkennen daran, daß die "Testzahl" 4 überhaupt nicht bearbeitet wurde, denn die Zahl 4 und alle ihre Vielfachen sind schon als Vielfache von

2 ausgenullt worden. Das Zwischenergebnis von STUFE 4 stellt dementsprechend auch keine Verbesserung gegenüber dem Zwischenergebnis von STUFE 3 dar.

Auch dieser Probelauf ist wieder ein Beispiel dafür, daß der DRUCKEZEILE-Befehl die äußersten Listenklammern wegläßt, wenn seine Eingabe eine Liste ist. Nach dem Wort ERGEBNIS: erscheint die Liste dagegen "in voller Pracht" mit allen Klammern.

Nun zur Hilfsfunktion UEBERSCHREIBE. Sie stellt unabhängig vom ERATOSTHENES-Problem eine nützliche Dienstfunktion (englisch: *utility*) dar.

```
PR UEBERSCHREIBE  :N  :L  :X
 WENN  :N > LAENGE :L   DANN  RUECKGABE  :L
 LOKAL  "L1
 SETZE  "L1  [ ]
 LOKAL  "I
 SETZE  "I  1
 SCHLEIFENBEGINN:
 WENN  :I < :N  DANN
         SETZE "L1  MITLETZTEM ERSTES :L :L1
 WENN  :I  =  :N  DANN
        SETZE "L1  ( SATZ  :L1  :X  OHNEERSTES :L )
       RUECKGABE :L1
 SETZE  "I  :I +1
 SETZE  "L  OHNEERSTES  :L
 WENN  :I > :N  DANN RUECKGABE :L1
 GEHE  "SCHLEIFENBEGINN
ENDE
```

Ein Beispiel im Direktbetrieb:

```
UEBERSCHREIBE  3  [ A B C D E F ]  "XERXES
ERGEBNIS:  [ A B XERXES D E F ]
```

Zu Übungszwecken ist es lehrreich, einmal eine endrekursive Version der Funktion ERATOSTHENES zu schreiben. Wir wollen sie im folgenden PRIMZAHLLISTE nennen. Bei ihrer Beschreibung können wir uns etwas kürzer fassen, da vieles schon im Zusammenhang mit der Funktion ERATOSTHENES besprochen wurde.

```
PR PRIMZAHLLISTE  :N
 LOKAL "KL
 SETZE "KL (MITERSTEM 0 OHNEERSTES KANDIDATENLISTE :N)
 PRIMZAHLLISTE.HP  2
 RUECKGABE  :KL
ENDE

PR PRIMZAHLLISTE.HP  :I
 WENN  :I * :I  >  :N  DANN  RUECKKEHR
 WENN NICHT? ( ELEMENT  :I  :KL )  =  0  DANN
            SETZE "KL  AUSNULLEN  :KL  :I
 ( DRUCKEZEILE  "STUFE  :I  ":  )  DRUCKEZEILE  :KL
 PRIMZAHLLISTE.HP  (:I + 1)
ENDE
```

Erläuterung der Prozedur PRIMZAHLLISTE: Diese Prozedur dient nur dazu, einige Vorbereitungen für den Aufruf der Hilfsprozedur PRIMZAHLLISTE.HP zu

erledigen und das dort ermittelte Ergebnis als Funktionswert zurückzugeben. PRIMZAHLLISTE definiert und bestückt die Hilfsvariable KL und ruft PRIMZAHLLISTE.HP mit dem Anfangswert 2 auf.

Die Prozedur PRIMZAHLLISTE.HP ist der eigentliche endrekursive Teil. Sie entspricht ungefähr der durch die Zeilen 7 bis 17 gegebenen Schleife in der Prozedur ERATOSTHENES. Die bearbeitete Kandidatenliste wird in der Variablen KL abgespeichert. KL ist zunächst eine lokale Variable von PRIMZAHLLISTE. Sie ist aber auch in der aufgerufenen Prozedur PRIMZAHLLISTE.HP bekannt.

Aufgabe 3.33: Warum darf :KL nicht zu einem Eingabeparameter der Prozedur PRIMZAHLLISTE.HP gemacht werden? Probiere aus, was dann passiert.

Da PRIMZAHLLISTE eine vollständig endrekursive Lösung darstellen sollte, ist es angebracht, auch die Hilfsprozedur AUSNULLEN in endrekursiver Form zu schreiben.

```
PR AUSNULLEN  :L  :I
 LOKAL  "L.ANFANG
 SETZE  "L.ANFANG  [ ]
 LOKAL  "L.ENDE
 SETZE  "L.ENDE  :L
 AUSNULLEN.HP  :I  (2 * :I)
 RUECKGABE  :L.ANFANG
ENDE

PR AUSNULLEN.HP  :I  :V
 WENN  :L.ENDE = [ ]  DANN RUECKKEHR
 PRUEFE  ( LAENGE  :L.ANFANG )  =  :V - 1
 WENNWAHR
      SETZE  "L.ANFANG  ( MITLETZTEM  0  :L.ANFANG )
      SETZE  "V  :V + :I
 WENNFALSCH
    SETZE "L.ANFANG (MITLETZTEM ERSTES :L.ENDE :L.ANFANG)
 SETZE "L.ENDE  OHNEERSTES :L.ENDE
 AUSNULLEN.HP  :I  :V
ENDE
```

Erläuterungen zur Prozedur AUSNULLEN: AUSNULLEN hat zwei Eingaben: die Liste :L und die Zahl :I. Ihre Aufgabe ist, alle Vielfachen von :I (außer :I selber) auszunullen. Nach dem bei endrekursiven Funktionen üblichen Muster haben wir auch hier wieder einen Vorbereitungs- und Aufruf-Teil (AUSNULLEN) und einen Teil, der die echte Arbeit erledigt (AUSNULLEN.HP). Der "Plan" sieht so aus, daß mit den beiden Hilfslisten :L.ANFANG und :L.ENDE gearbeitet wird. Zu Beginn ist :L.ANFANG leer und :L.ENDE gleich :L.

Die Hilfsprozedur AUSNULLEN.HP hat zwei Eingaben: die aktuelle "Testzahl" :I und ihre Vielfachen :V. Im wesentlichen werden die Elemente von :L.ENDE eins nach dem anderen nach :L.ANFANG "geschoben". Jeweils das erste Element von :L.ENDE wird als letztes Element an :L.ANFANG angehängt (und aus :L.ENDE entfernt). Nur wenn die Stelle in :L.ANFANG gerade ein Vielfaches von :V ist, wird das erste Element von :L.ENDE fallengelassen und stattdessen eine Null an :L.ANFANG angehängt.

Mit dem PRUEFE-Befehl kann man lange WENN ... DANN ... SONST - Schachtelungen vermeiden. Ohne diesen Befehl müßte man an Stelle der mit den Worten

PRUEFE, WENNWAHR (kurz: WW) und WENNFALSCH (kurz: WF) beginnenden Zeilen in AUSNULLEN.HP den folgenden "Bandwurm" in einer einzigen logischen Zeile schreiben:

```
WENN ( LAENGE :L.ANFANG ) = :V - 1  DANN _________
___ SETZE  "L.ANFANG (MITERSTEM 0 :L.ANFANG ) ___
___ SETZE  "V  :V + :I ____________________________
___ SONST  SETZE "L.ANFANG ________________________
___        ( MITLETZTEM ERSTES :L.ENDE :L.ANFANG )
```

Aufgabe 3.34: Schreibe eine Logo-Funktion NULLEN.RAUS, die alle in einer Liste enthaltenen Nullen entfernt. Der Aufruf NULLEN.RAUS PRIMZAHLLISTE 100 liefert als Ergebnis dann eine Liste mit den Primzahlen zwischen 1 und 100.

Bei den Laufzeiten weisen die Funktionen ERATOSTHENES und PRIMZAHL-LISTE deutliche Unterschiede auf. Der Aufruf ERATOSTHENES 50 benötigt 2 Minuten 23 Sekunden, der von PRIMZAHLLISTE 50 dagegen nur 28 Sekunden. Dies liegt daran, daß die Hilfsfunktion AUSNULLEN dem Siebverfahren besser angepaßt ist als die Hilfsfunktion UEBERSCHREIBE. Die Funktion UEBER-SCHREIBE ist deswegen so langsam, weil es im Commodore Logo keinen schreibenden Direktzugriff auf das :N-te Element einer Liste gibt. (Einen lesenden Direktzugriff ermöglicht der Grundbefehl ELEMENT, wie wir gesehen haben).

Aufgabe 3.35: Der lesende und schreibende Direktzugriff auf bestimmte Elemente ist üblicherweise bei **Feldern** (englisch: *arrays*) möglich. Falls du eine Programmiersprache mit Feldern hast, in der du dich einigermaßen auskennst, dann entwickle eine "Feld"-Version für das Sieb des Eratosthenes. Moderne Logo-Versionen auf etwas größeren Personal Computern verfügen im allgemeinen auch über Felder.

Aufgabe 3.36: Schreibe eine endrekursive Version der Prozedur UEBERSCHREIBE.

Aufgabe 3.37: *Primzahl-"Bootstrapping"* (ein kleines Projekt): Wir wollen eine Liste anlegen, die ein "Anfangsstück" der Primzahlen, also alle Primzahlen bis zu einer bestimmten oberen Schranke, enthält. Im Prinzip könnten wir das zum Beispiel mit der Prozedur ERATOSTHENES oder PRIMZAHLLISTE machen, wobei wir zum Schluß alle Nullen aus der Ergebnisliste hinauswerfen müssen. Aber dieses Verfahren ist sehr langsam. Wir wollen anders vorgehen. Zunächst verschaffen wir uns mit Hilfe der Funktion ERATOSTHENES (oder auch "von Hand") unter dem globalen Namen PRIMZAHLEN eine kleine, aufsteigend geordnete Anfangsliste von Primzahlen; zum Beispiel alle Primzahlen bis zur Zahl 7:

```
WERT "PRIMZAHLEN = [ 2 3 5 7 ]
```

Dann testen wir, beginnend bei 9, in Zweierschritten vorwärtsschreitend, alle ungeraden Zahlen. Wenn wir auf eine Primzahl stoßen, dann nehmen wir sie als letztes Element in unsere Liste der Primzahlen auf. Dadurch wird die Liste der Primzahlen immer größer.

Diese Liste der Primzahlen können wir auch gleich für einen sehr viel schnelleren Primzahltest verwenden. Bisher haben wir bei Primzahlkandidaten geprüft, ob sie durch *irgendeine (kleinere) Zahl* teilbar sind. Um zu testen, ob eine Zahl eine Primzahl ist, müssen wir aber nur prüfen, ob sie durch eine *Primzahl* teilbar ist. Und für diesen Test können wir die laufend erweiterte Primzahlliste heranziehen.

Ein Beispiel: Um zu testen, ob 269 eine Primzahl ist, brauchen wir diese Zahl nur durch 2, 3, 5, 7, 11, 13 und 17 zu teilen, nicht jedoch durch 4, 6, 8, 9, 10, 12, 14, 15 und 16. Bei größeren Testzahlen wird noch viel mehr eingespart.

Das hier beschriebene Verfahren erinnert an die Geschichte des Barons von Münchhausen, der sich am eigenen Schopf aus dem Sumpf gezogen hat. Das Wort *bootstrapping* hat im Englischen ungefähr diese Bedeutung. Man nennt derartige Verfahren deshalb Bootstrap-Verfahren.

Führe das eben beschriebene Verfahren aus. Ein Hinweis: Die Primzahlliste PRIMZAHLEN kann dabei als globale Variable geführt werden. Wenn überhaupt eine Variable es verdient, global geführt zu werden, dann die Variable, welche die Primzahlen enthält.

Der Lauf der Primzahl-Bootstrap-Programms kann übrigens jederzeit (mit Control-G) unterbrochen werden. Wenn man die bis dahin vorliegende Primzahlliste sichert (z.B. auf einer Diskette abspeichert), kann man den Bootstrap-Lauf beim nächsten Mal mit dieser Liste als Anfangsstück fortsetzen. Hier ist ein solches Anfangsstück der Primzahlliste:

```
[   2    3    5    7   11   13   17   19   23   29
   31   37   41   43   47   53   59   61   67   71
   73   79   83   89   97  101  103  107  109  113
  127  131  137  139  149  151  157  163  167  173
  179  181  191  193  197  199  211  223  227  229
  233  239  241  251  257  263  269  271  277  281
  283  293  307  311  313  317  331  337  347  349
  353  359  367  373  379  383  389  397  401  409
  419  421  431  433  439  443  449  457  461  463
  467  479  487  491  499  503  509  521  523  541
  547  557  563  569  571  577  587  593  599  601
  607  613  617  619  631  641  643  647  653  659
  661  673  677  683  691  701  709  719  727  733
  739  743  751  757  761  769  773  787  797  809
  811  821  823  827  829  839  853  857  859  863
  877  881  883  887  907  911  919  929  937  941
  947  953  967  971  977  983  991  997 1009 1013
 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069
 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151
 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223
 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291
 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373
 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451
 1453 1459 1471 1481 1483 1487 1489 1493 1499 1511
 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583
 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657
 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733
 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811
 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889
 1901 1907 1913 1931 1933 1949 1951 1973 1979 1987
 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053
 2063 2069 2081 2083 2087 2089 2099 2111 2113 2129
 2131 2137 2141 2143 2153 2161 2179 2203 2207 2213
 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287
 2293 2297 2309 2311 2333 2339 2341 2347 2351 2357
 2371 2377 2381 2383 2389 2393 2399 2411 2417 2423
```

```
2437 2441 2447 2459 2467 2473 2477 2503 2521 2531
2539 2543 2549 2551 2557 2579 2591 2593 2609 2617
2621 2633 2647 2657 2659 2663 2671 2677 2683 2687
2689 2693 2699 2707 2711 2713 2719 2729 2731 2741
2749 2753 2767 2777 2789 2791 2797 2801 2803 2819
2833 2837 2843 2851 2857 2861 2879 2887 2897 2903
2909 2917 2927 2939 2953 2957 2963 2969 2971 2999
3001 3011 3019 3023 3037 3041 3049 3061 3067 3079
3083 3089 3109 3119 3121 3137 3163 3167 3169 3181
3187 3191 3203 3209 3217 3221 3229 3251 3253 3257
3259 3271 3299 3301 3307 3313 3319 3323 3329 3331
3343 3347 3359 3361 3371 3373 3389 3391 3407 3413
3433 3449 3457 3461 3463 3467 3469 3491 3499 3511
3517 3527 3529 3533 3539 3541 3547 3557 3559 3571
3581 3583 3593 3607 3613 3617 3623 3631 3637 3643
3659 3671 3673 3677 3691 3697 3701 3709 3719 3727
3733 3739 3761 3767 3769 3779 3793 3797 3803 3821
3823 3833 3847 3851 3853 3863 3877 3881 3889 3907
3911 3917 3919 3923 3929 3931 3943 3947 3967 3989
4001 4003 4007 4013 4019 4021 4027 4049 4051 4057
4073 4079 4091 4093 4099 4111 4127 4129 4133 4139
4153 4157 4159 4177 4201 4211 4217 4219 4229 4231
4241 4243 4253 4259 4261 4271 4273 4283 4289 4297
4327 4337 4339 4349 4357 4363 4373 4391 4397 4409
4421 4423 4441 4447 4451 4457 4463 4481 4483 4493
4507 4513 4517 4519 4523 4547 4549 4561 4567 4583
4591 4597 4603 4621 4637 4639 4643 4649 4651 4657
4663 4673 4679 4691 4703 4721 4723 4729 4733 4751
4759 4783 4787 4789 4793 4799 4801 4813 4817 4831
4861 4871 4877 4889 4903 4909 4919 4931 4933 4937
4943 4951 4957 4967 4969 4973 4987 4993 4999 5003
5009 5011 5021 5023 5039 5051 5059 5077 5081 5087
5099 5101 5107 5113 ]
```

Aufgabe 3.38: Schreibe eine Funktion PRIMZAHL :I, welche die :I-te Primzahl als Funktionswert zurückgibt. Zum Beispiel:
PRIMZAHL 5
ERGEBNIS: 11

Aufgabe 3.39: Die **Goldbachsche Vermutung** (nach *Chr. Goldbach*, 1690-1764) besagt, daß man jede gerade Zahl größer oder gleich 6 als Summe von zwei Primzahlen schreiben kann. Zum Beispiel ist $12 = 5 + 7$ oder $24 = 7 + 17 = 11 + 13$. Es kann also u.U. auch mehrere solche Summendarstellungen geben. Überprüfe die Goldbachsche Vermutung mit Hilfe der Primzahlliste aus dem Primzahl-Bootstrap-Lauf an einigen Beispielen
(a) von Hand,
(b) mit Hilfe eines Programms.

Aufgabe 3.40: (a) Schreibe eine Funktion ZWEI.HOCH :N, mit der man die Zweierpotenzen ermitteln kann.
(b) Schreibe eine Funktion HOCH :X :N, mit der man die :N-te Potenz einer beliebigen Zahl :X ermitteln kann.

Hier ist eine schnelle Lösung für das Potenzieren:

```
PR POT  :X  :N
 LOKAL  "Y
 SETZE  "Y  1
 POT.HP  :X  :N
 RUECKGABE  :Y
ENDE

PR POT.HP  :X  :N
 WENN  :N = 0  DANN RUECKKEHR
 PRUEFE  GERADE?  :N
 WENNWAHR  SETZE  "X  :X * :X  SETZE  "N  ( DIV :N 2 )
 WENNFALSCH SETZE "Y  :Y * :X  SETZE  "N  ( :N - 1 )
 POT.HP  :X  :N
ENDE

PR GERADE?  :N
 RUECKGABE  ( REST  :N  2 )  =  0
ENDE
```

Die folgende Tabelle gibt das "Kurzprotokoll" des Aufrufs POT 3 15 wieder:

:Y	:X	:N	:Y * (:X hoch :N)
1	3	15	$1 * 3^{15}$
3	3	14	$3 * 3^{14}$
3	9	7	$3 * 9^{7}$
27	9	6	$27 * 9^{6}$
27	81	3	$27 * 81^{3}$
2187	81	2	$2187 * 81^{2}$
2187	6561	1	$2187 * 6561^{1}$
14348907	6561	0	$14348907 * 6561^{0}$

Tabelle 3.4: Kurzprotokoll des Aufrufs POT 3 15

Aufgabe 3.41: (a) Erweitere die Prozedur POT.HP durch eine weitere Programmzeile zu Demonstrationszwecken so, daß unmittelbar nach dem Aufruf die Werte der Variablen Y, X und N ausgedruckt werden. Verfolge einige Demonstrationsläufe.
(b) Begründe die Tatsache, daß sich während des Laufes von POT.HP der Wert des Terms :Y * (:X hoch :N) nicht verändert.
(c) Gib den Wert der Variablen Y nach Ablauf der Prozedur POT.HP an.
(d) Die schnelle Version von POT / POT.HP ist besonders vorteilhaft, wenn :N eine Zweierpotenz ist. Begründe dies. Gib für diese besonders günstigen Fälle an, wie viele Durchläufe von POT.HP notwendig sind, bis das Ergebnis ermittelt ist.
(e) Stelle die Druckbefehle aus Aufgabenteil (a) "kalt", indem du ein Semikolon an den Anfang der entsprechenden Zeile schreibst. Diese Zeile wird dann als Kommentarzeile gewertet. Wenn du die Ausdruckbefehle wieder "einschalten" möchtest, brauchst du nur das Semikolon zu entfernen.

Aufgabe 3.42: Der französiche Mathematiker *Mersenne* (1588 - 1648) untersuchte Zahlen der Art (ZWEI.HOCH :P) - 1, wobei :P eine Primzahl ist. (Solche Zahlen werden seitdem **Mersennesche Zahlen** genannt). Er stellte die Vermutung auf, daß alle diese Zahlen Primzahlen sind. So ist zum Beispiel (ZWEI.HOCH 5) - 1 = 31 eine Primzahl. Überprüfe die Mersennesche Vermutung.

Aufgabe 3.43: Abgesehen von der Primzahl 2 sind alle weiteren Primzahlen ungerade; sie müssen sich also um mindestens 2 unterscheiden. Tatsächlich gibt es eine Reihe von Primzahlpaaren, deren Differenz genau 2 ist; zum Beispiel 11 und 13, 17 und 19, 41 und 43 usw. Solche Primzahlpaare heißen *Primzahlzwillinge*. Suche weitere Primzahlzwillinge.

Aufgabe 3.44: (a) Schreibe eine Prozedur ZAHLENTABELLE :SPALTENZAHL, mit der du die natürlichen Zahlen nacheinander in der angegebenen Spaltenzahl ausdruckst. Ein Beispiel:

```
ZAHLENTABELLE  12

  1  (2) (3)  4  (5)  6  (7)  8   9  10 (11) 12
(13) 14  15  16 (17) 18 (19) 20  21  22 (23) 24
 25  26  27  28 (29) 30 (31) 32  33  34  35  36
(37) 38  39  40 (41) 42 (43) 44  45  46 (47) 48
 49  50  51  52 (53) 54  55  56  57  58 (59) 60
(61) 62  63  64  65  66 (67) 68  69  70 (71) 72
(73) 74  75  76  77  78 (79) 80  81  82 (83) 84
 85  86  87  88 (89) 90  91  92  93  94  95  96

...
```

(b) Beobachte, wie sich die Primzahlen auf die einzelnen Spalten verteilen.
(c) Begründe, warum in der zweiten, dritten, vierten, sechsten, achten, neunten, zehnten und zwölften Spalte ab Zeile 2 keine Primzahlen mehr zu finden sind.
(d) Welche Eigenschaften haben die "Spaltenzahlen" 1, 5, 7 und 11 bezogen auf die Zahl 12?

Aufgabe 3.45: Der kleine Fritz spielt gern mit Zahlen. Er stellt dabei folgendes fest:

```
2 * 3  +  1  =  7                    (Primzahl)
2 * 3 * 5  + 1  =  31                (Primzahl)
2 * 3 * 5 * 7  + 1  =  211           (Primzahl)
...
```

Schreibe die nächste Zeile nach Fritzens Bildungsgesetz auf. Fritz vermutet, daß immer wieder Primzahlen herauskommen. Überprüfe diese Vermutung.

Aufgabe 3.46: Einige der Prozeduren dieses Abschnitts liegen in verschiedenen Versionen vor. Entscheide dich jeweils für eine lauffähige Arbeitsversion. Füge diese Arbeitsversionen den bisherigen Prozeduren der Teilbarkeits-Datei hinzu. (Du kannst dabei entsprechend vorgehen wie in Aufgabe 3.24).

3.4 Der größte gemeinsame Teiler

Aufgabe 3.47: Ein rechteckiger Papierbogen ist 136 cm lang und 60 cm breit. Er soll in gleich große Quadrate zerschnitten werden, ohne daß ein Rest übrig bleibt.

Man kann sich vorstellen, daß der Papierbogen mit den Quadraten "gepflastert" ist, wie in der folgenden Abbildung:

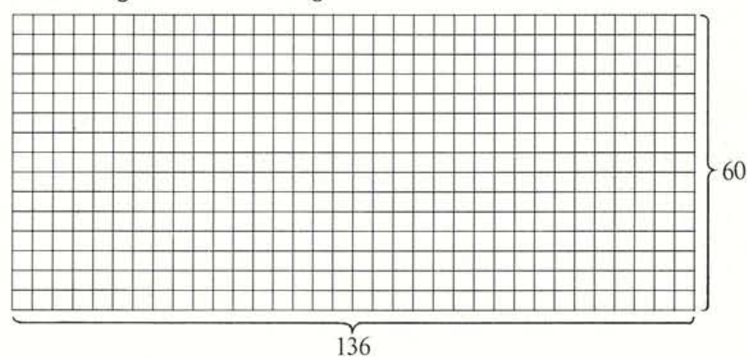


Bild 3.1: Pflasterung mit Quadraten

Es gibt verschiedene Möglichkeiten, das Rechteck zu zerschneiden. Wenn man möglichst viele Quadrate haben will, dann muß man nur ihre Seitenlänge klein wählen; zum Beispiel 1 cm oder noch kleiner.

Aufgabe 3.48: Wie viele Quadrate erhält man bei einer Seitenlänge von 1 cm; wie viele bei einer Seitenlängen von 0.5 cm ?

Aufgabe 3.49: Welches ist die größtmögliche Seitenlänge?

Aus Bild 3.1 geht hervor, daß die Seitenlänge des Quadrates in jeder der Rechtecksseiten "aufgehen" muß. Das heißt, sie muß jede der Rechtecksseiten teilen.

Aufgabe 3.50: (a) Ermittle die Teiler der Zahl 136 und der Zahl 60.
(b) Bestimme die gemeinsamen Teiler.
(c) Bestimme den größten gemeinsamen Teiler.

Merke: Die Seitenlänge des größten Quadrates, mit dem ein Rechteck der Seitenlängen a und b gepflastert werden kann, ist der **größte gemeinsame Teiler (ggT)** bzw. das *größte gemeinsame Maß* von a und b.

Der größte gemeinsame Teiler zweier natürlicher Zahlen wird in der Mathematik häufig benötigt; zum Beispiel beim Kürzen eines Bruches. Im obigen Beispiel kommt er in der Form des größten gemeinsamen Maßes zweier Strecken vor. Wir wollen nun verschiedene Möglichkeiten zur Bestimmung des größten gemeinsamen Teilers kennenlernen.

Gehen wir einmal ganz naiv vor. Zur Bestimmung der Teiler von 136 können wir unsere TEILER-Funktion verwenden:

```
TEILER 136
ERGEBNIS: [ 1 2 4 8 17 34 68 136 ]
```

TEILER 60
ERGEBNIS: [1 2 3 4 5 6 9 10 12 15 20 60]

Die gemeinsamen Teiler sind 1, 2 und 4; der größte gemeinsame Teiler ist 4.

Was wir hier "von Hand" durchgeführt haben, können wir auch von Logo ausführen lassen. Wir benötigen dazu offenbar eine Funktion, die uns aus zwei Listen die gemeinsamen Elemente herausholt, und eine weitere Funktion, die aus einer Liste die größte Zahl "herausfischt".

```
PR GEMEINSAME.ELEMENTE  :L1  :L2
 LOKAL  "D
 SETZE  "D  [ ]
 WENN  :L2 = [ ]  DANN RUECKGABE :D
 MARKIERUNG1:
 WENN  :L1 = [ ]  DANN RUECKGABE :D
 WENN  EL? (ERSTES :L1) :L2  DANN __________
 ___    SETZE  "D  ( MITLETZTEM  ERSTES :L1 )  :D
 SETZE  "L1  OHNEERSTES  :L1
 GEHE "MARKIERUNG1
ENDE
```

Erläuterungen zur Prozedur GEMEINSAME.ELEMENTE: D ist der Name einer Hilfsliste, in der die gemeinsamen Elemente gespeichert werden. Am Anfang ist D die leere Liste. Wenn eine der beiden Ausgangslisten :L1 bzw. :L2 leer ist, kann es keine weiteren gemeinsamen Elemente geben, und wir sind fertig. Die Logo-Prüffunktion EL? (Kurzform für ELEMENT? - die Langform gibt es aber nicht als Grundwort) erwartet zwei Eingaben; ein beliebiges Objekt und eine Liste. Sie überprüft, ob das angegebene Objekt ein Element der Liste ist oder nicht. Hierzu einige Beispiele:

EL? 3 [A 1 B 2 C 3 D 4]
ERGEBNIS: WAHR

EL? 3 [333 BEI ISSUS KEILEREI]
ERGEBNIS: FALSCH

EL? "XYZ [A B C X Y Z]
ERGEBNIS: FALSCH

EL? [R S] [1 2 Q W [R S] Z]
ERGEBNIS: WAHR

EL? [R S] [E R S T E S]
ERGEBNIS: FALSCH

Der Befehl SETZE "D (MITLETZTEM ERSTES :L1) :D sorgt schließlich dafür, daß Elemente von :L1, die auch in :L2 vorkommen, in die Liste D der gemeinsamen Elemente aufgenommen werden.

Wir wollen uns zum besseren Verständnis ein Ablaufprotokoll anschauen.

GEMEINSAME.ELEMENTE [1 2 3 6 9 18] [1 3 5 15]

Schleifen-durchlauf	Wert von L1 vorher	Wert von L1 nachher	Wert von D vorher	Wert von D nachher
1	[1 2 3 6 9 18]	[2 3 6 9 18]	[ ]	[1]
2	[2 3 6 9 18]	[3 6 9 18]	[1]	[1]
3	[3 6 9 18]	[6 9 18]	[1]	[1 3]
4	[6 9 18]	[9 18]	[1 3]	[1 3]
5	[9 18]	[18]	[1 3]	[1 3]
6	[18]	[ ]	[1 3]	[1 3]

Rückgabewert der Funktion GEMEINSAME.ELEMENTE

Tabelle 3.5

Die Funktion GEMEINSAME.ELEMENTE funktioniert mit beliebigen Listen als Eingabe und stellt deshalb ein nützliches allgemeines Dienstprogramm dar.

GEMEINSAME.ELEMENTE [A B C 17 34 51] [B C D 51 68 Z]
ERGEBNIS: [B C 51]

GEMEINSAME.ELEMENTE [X Y Z] [1 2 3 4 5 6]
ERGEBNIS: [ ]

Aufgabe 3.51: Schreibe eine Funktion ENTHALTEN? :L1 :L2 , mit der du prüfen kannst, ob jedes Element der Liste :L1 in der Liste :L2 enthalten ist.

Aufgabe 3.52: Schreibe eine endrekursive Version der Funktion GEMEINSAME.ELEMENTE.

Aufgabe 3.53: Schreibe eine Funktion GEMEINSAME.TEILER :A :B

Da unsere Teilerlisten der Größe nach geordnet sind, ist der größte gemeinsame Teiler das letzte Element in der Liste der gemeinsamen Teiler.

```
PR GROESSTER.GEMEINSAMER.TEILER  :A  :B
 RUECKGABE  LETZTES
       GEMEINSAME.ELEMENTE (TEILER :A) (TEILER :B)
ENDE
```

GROESSTER.GEMEINSAMER.TEILER 12 18
ERGEBNIS: 6

Aufgabe 3.54: Verwende die schnellere TEILER-Funktion in GROESSTER.GEMEINSAMER.TEILER.

Aufgabe 3.55: Schreibe eine Dienstfunktion GROESSTE.ZAHL :L, deren Funktionswert die größte in der Liste :L enthaltene Zahl ist.

Die Funktion GROESSTER.GEMEINSAMER.TEILER ist nicht besonders schnell. Aber sie tut ihren Dienst. Und sie ist völlig natürlich aufgebaut. Außerdem haben wir dabei eine ganze Anzahl nützlicher Dienstfunktionen kennengelernt, die man sowieso des öfteren braucht.

Es gibt aber sehr viel bessere Möglichkeiten, den größten gemeinsamen Teiler zweier Zahlen zu bestimmen. Das folgende Verfahren stammt von dem griechischen Mathematiker *Euklid von Alexandria* (etwa 365 - 300 v. Chr.)

Dem euklidischen Verfahren (**Euklidischer Algorithmus**) liegt die folgende einfache Beobachtung zugrunde: Jeder gemeinsame Teiler der Zahlen a und b ist auch ein Teiler ihrer Differenz a-b. Um dies einzusehen, stellen wir uns die Zahlen a und b als Strecken vor. Ein Beispiel mit a = 18 und b = 14:

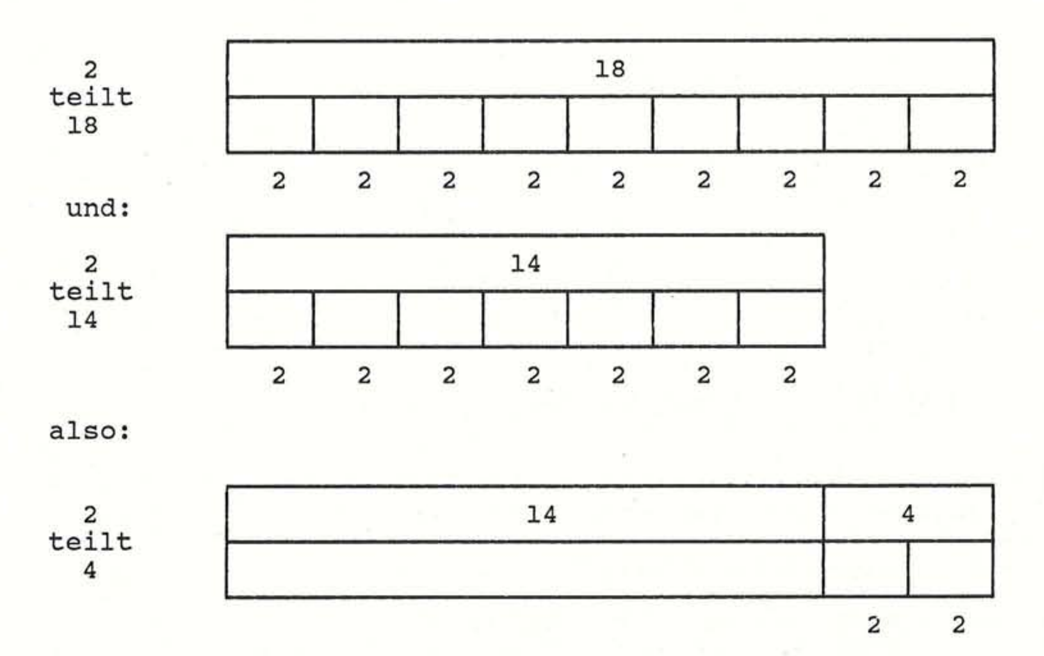


Bild 3.2: Die Strecke 2 ist ein *gemeinsames Maß* der Strecken 18 und 14.

Die Strecke 2 geht ganz in der Strecke 18 auf; ebenso in der Strecke 14. Also muß sie auch in der Differenzstrecke 4 aufgehen.

Wir wollen diese Beobachtung nun auf den größten gemeinsamen Teiler von 136 und 60 anwenden. Wenn uns diese Aufgabe als zu schwer erscheint, können wir uns das Leben dadurch leichter machen, daß wir den größten gemeinsamen Teiler der Zahlen 136 - 60 und 60, also von 76 und 60 zu finden versuchen. Aber auch dieses Problem läßt sich weiter vereinfachen. Insgesamt erhalten wir schrittweise die folgenden einfacheren Probleme:

```
größter gemeinsamer Teiler von 136 und 60 =
größter gemeinsamer Teiler von  76 und 60 =
größter gemeinsamer Teiler von  16 und 60 =
größter gemeinsamer Teiler von  16 und 44 =
größter gemeinsamer Teiler von  16 und 28 =
größter gemeinsamer Teiler von  16 und 12 =
größter gemeinsamer Teiler von   4 und 12 =
größter gemeinsamer Teiler von   4 und  8 =
größter gemeinsamer Teiler von   4 und  4 =  4
```

Dieses Verfahren, daß man immer die kleinere von der größeren Zahl abzieht, bezeichnet man auch als **Wechselwegnahme**.

Geometrisch können wir uns das Verfahren so veranschaulichen:

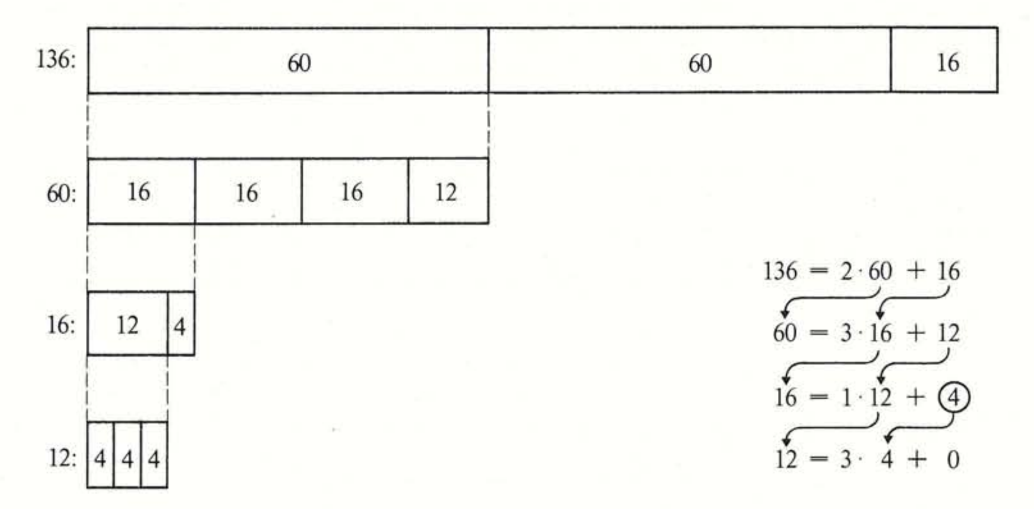


Bild 3.3: Der Euklidische Algorithmus

Nach diesen Vorbereitungen können wir nun leicht eine Logo-Funktion zur Bestimmung des größten gemeinsamen Teilers schreiben.

```
PR GGT.SUBTRAKTIONSFORM  :A  :B
 MARKIERUNG1:
 WENN :A = :B  DANN RUECKGABE  :A
 WENN :A > :B  DANN SETZE  "A  (:A - :B)
 WENN :B > :A  DANN SETZE  "B  (:B - :A)
 GEHE "MARKIERUNG1
ENDE
```

Im Falle :A=0 oder :B=0 läuft GGT.SUBTRAKTIONSFORM noch nicht richtig. Dies wird im Zusammenhang mit anderen Verbesserungen gleich noch korrigiert.

Aufgabe 3.56: Schreibe eine endrekursive Version der Funktion GGT.SUBTRAKTIONSFORM.

Der Euklidische Algorithmus ist ein sehr viel schnelleres Verfahren als unser erster Versuch mit der Funktion GROESSTER.GEMEINSAMER.TEILER. Aber auch dieser Algorithmus läßt sich noch erheblich verbessern.

Nehmen wir einmal an, wir wollen den größten gemeinsamen Teiler der Zahlen 2098757 und 17 mit dem Subtraktionsalgorithmus berechnen. Dann wird nach dem obigen Algorithmus 123456 mal die 17 von der Zahl 2098757 abgezogen, bis wir eine Differenz unterhalb von 17 erhalten (in diesem Fall: die Differenz 5). Diese vielen Subtraktionen benötigen sehr viel Zeit. Ein klein wenig Nachdenken sagt uns aber, daß die 5 gerade der Divisionsrest bei der Division von 2098757 durch 17 ist. Denn der Divisionsrest ist doch gerade die Zahl, die übrig bleibt, wenn man so viele "Siebzehner" wie möglich von 2098757 abzieht. Diesen Divisionsrest erhalten wir aber sehr viel schneller, wenn wir die Logo-Grundfunktion REST benutzen:

REST 2098757 17
ERGEBNIS: 5

Wir bauen diese Verbesserung sofort in unsere obige GGT-Funktion ein:

```
1:   PR GGT.MIT.REST  :A  :B
2:    MARKIERUNG1:
3:    WENN :A = :B  DANN RUECKGABE  :A
4:    WENN :A > :B  DANN SETZE  "A  ( REST  :A  :B )
5:    WENN :B > :A  DANN SETZE  "B  ( REST  :B  :A )
6:    GEHE "MARKIERUNG1
7:   ENDE
```

Beim Lauf dieser Version erhalten wir aber die Fehlermeldung "DIVISION DURCH NULL!". Dies ist natürlich verboten. Aber wo wird denn hier durch Null dividiert? Schauen wir uns den Lauf von GGT.MIT.REST einmal aufmerksam im abgekürzten Protokoll-Modus an. Dazu bauen wir zwischen Zeile 2 und 3 den Druckbefehl (DZ :A :B) ein. Ein Probelauf:

GGT.MIT.REST 18 12
18 12
6 0
DIVISION DURCH NULL!

Wie ist das passiert? Wir begannen mit :A = 18 und :B = 12. In Zeile 4 (alte Numerierung) wurde dann der Wert von A auf 6 (= REST 18 12) und in Zeile 5 der Wert von B auf 0 (= REST 12 6) gesetzt. Im nächsten Schleifendurchlauf wäre dann in Zeile 4 REST 6 0 zu berechnen gewesen; und dabei trat die Fehlermeldung auf.

Erinnerst du dich an den Anfang von Abschnitt 3.1? Wir haben dort gesehen, daß die REST-Funktion eng mit der Ganzzahldivision zusammenhängt. Probiere folgendes aus:

DIV 6 0
DIVISION DURCH NULL!
REST 6 0
DIVISION DURCH NULL!

6 / 0
DIVISION DURCH NULL!

Wir haben die Aussage "a teilt b" bisher so verstanden, daß die Division von b durch a den Rest Null ergibt. Dies ist zwar nicht falsch, um einige Sonderfälle mit der Null besser in den Griff zu bekommen, ist die folgende Definition günstiger:

Die Zahl a teilt die Zahl b, wenn es eine natürliche Zahl n gibt mit: a * n = b.

Die Aufgabe "6 geteilt durch 0" ist zwar mathematisch nicht sinnvoll, aber die Aussage "0 ist kein Teiler von 6" ist dennoch richtig, denn es gibt keine natürliche Zahl n mit der Eigenschaft 0 * n = 6. Auch die Aussage "0 ist ein Teiler von 0" ist wahr, denn 0 * m = 0 ist für jede Zahl m richtig. Wir sollten deshalb auch die Prüffunktion TEILT? aus Abschnitt 3.1 dieser verbesserten Definition anpassen:

```
PR TEILT?  :A  :B
 WENN :B = 0  DANN RUECKGABE  "WAHR
 WENN :A = 0  DANN RUECKGABE  "FALSCH
 RUECKGABE  ( REST  :B  :A )  =  0
ENDE
```

Jetzt können wir die obige Fassung der Funktion GGT.MIT.REST reparieren. Denn die verhängnisvolle Zahl Null kam dadurch zustande, daß die eine Zahl (im obigen Beispiel die 6) die andere teilte (im obigen Beispiel die 12). In einem solchen Fall können wir aber sofort sagen, was der größte gemeinsame Teiler ist.

Merke: Wenn die Zahl a ein Teiler der Zahl b ist, dann ist der größte gemeinsame Teiler von a und b gleich a selbst. Zum Beispiel: ggT(6 , 12) = 6.

Hier ist eine korrekte Version unserer Prozedur:

```
PR GGT.MIT.REST  :A  :B
 MARKIERUNG1:
 WENN  TEILT?  :A  :B  DANN RUECKGABE :A
 WENN  TEILT?  :B  :A  DANN RUECKGABE :B
 WENN :A > :B  DANN SETZE  "A  ( REST  :A  :B )
 WENN :B > :A  DANN SETZE  "B  ( REST  :B  :A )
 GEHE "MARKIERUNG1
ENDE
```

Während die vorige Version GGT.SUBTRAKTIONSFORM bei ungünstigen Zahlen minutenlang oder gar stundenlang subtrahierte, liefert uns die REST-Version das Ergebnis schlagartig. Man sieht: es lohnt sich, den Problemen mit etwas mehr Mathematik auf den Leib zu rücken.

Die Version mit REST ist so *effizient*, d.h. sie benötigt so wenige Schritte, daß in diesem Fall selbst eine voll rekursive Version der Prozedur möglich ist, ohne daß wir allzu schnell die Fehlermeldung "SPEICHER VOLL" bekommen.

```
PR GGT  :A  :B
 WENN  TEILT?  :A  :B  DANN RUECKGABE :A
 WENN  TEILT?  :B  :A  DANN RUECKGABE :B
 WENN :A > :B  DANN RUECKGABE  GGT ( REST  :A  :B ) :B
 WENN :B > :A  DANN RUECKGABE  GGT :A  ( REST  :B  :A )
ENDE
```

Aufgabe 3.57: (a) Verfolge die letzte Version aufmerksam im (eingebauten) Protokollmodus.
(b) Baue unmittelbar hinter die erste Zeile den Druckbefehl (DRUCKEZEILE :A :B) ein und beobachte den Lauf der Prozedur in diesem abgekürzten Protokollmodus. Entferne danach den Druckbefehl wieder.

Aufgabe 3.58: Man kann den größten gemeinsamen Teiler auch über die Primfaktorzerlegung zweier Zahlen ermitteln. Dieses Verfahren wird am besten durch ein Beispiel verdeutlicht.

```
a  =  14700  =  2 * 2 * 3 * 5 * 5 * 7 * 7
b  =   3234  =  2 *     3 *         7 * 7 * 11
----------------------------------------------
ggT(a , b)   =  2 *     3 *         7 * 7       =  294
```

Schreibe eine Funktion GGT.PFZ, die den größten gemeinsamen Teiler zweier Zahlen über ihre Primfaktorzerlegung ermittelt. Hinweis: Die Funktion PRIMFAKTORZERLEGUNG aus Abschnitt 3.3 kann dabei verwendet werden. Dagegen ist die Funktion GEMEINSAME.ELEMENTE in der vorliegenden Form für dieses Problem noch nicht brauchbar. Begründe dies. Schreibe eine geeignete Funktion GEMEINSAME.ELEMENTE.NEU.

Aufgabe 3.59: Vergleiche die Laufzeiten aller GGT-Versionen, die wir besprochen haben. Betrachte verschiedene Werte von A und B. Lege eine Tabelle an.

Aufgabe 3.60: Beobachte, was passiert, wenn eine der Eingaben negativ ist. Baue die Funktionen so aus, daß man auch negative Zahlen eingeben kann. Verfahre dabei wie im folgenden Beispiel:

ggT(12 , -18) = ggT(-12 , 18) = ggT(-12 , -18) = 6

(Man kann diese Form des GGT beim Kürzen von Brüchen gut gebrauchen).

Aufgabe 3.61: Angenommen, ein Zahlenpaar zwischen 1 und 1000 wird in zufälliger Weise ausgewählt. Schätze die Chancen dafür, daß die Zahlen teilerfremd sind (d.h. den größten gemeinsamen Teiler 1 haben).

Aufgabe 3.62: Schreibe eine Funktion, die systematisch alle Zahlenpaare zwischen 1 und 1000 durchprüft und so den Anteil der teilerfremden Paare ermittelt.

Wenn man über einen Computer verfügt, dann kann man für viele Probleme, die man sonst vielleicht gar nicht lösen kann, eine angenäherte Antwort durch Anwendung eines **Simulationsverfahrens** (vgl. Kapitel 6) gewinnen. Wir wollen am Beispiel des letzten Problems einmal zeigen, wie so etwas funktioniert.

Logo verfügt über das Grundwort ZUFALLSZAHL (kurz ZZ). Dieser Befehl erwartet als Eingabe eine natürliche Zahl. Der Aufruf ZUFALLSZAHL :N liefert dann eine zufällig ausgewählte Zahl zwischen 0 und :N-1 (jeweils einschließlich der Grenzen). Wie es der Computer, bei dem ja eigentlich alles streng logisch vorgehen sollte, fertig bringt, Zahlen zufällig auszuwählen, wollen wir hier nicht weiter verfolgen. Wir sehen es einfach als gegeben an, daß er das kann.

Hier ein kleines Experiment mit den Zufallszahlen:

```
ZZ 100
ERGEBNIS: 27

WH 5 [ DZ ZZ 1000 ]
857
12
129
650
289
```

Wenn der Befehl ZUFALLSZAHL 1000 Zahlen zwischen 0 und 999 liefert, dann ergibt der Ausdruck (1 + ZUFALLSZAHL 1000) bei jedem Aufruf eine Zahl zwischen 1 und 1000. Wir können diese Zahlen direkt in die GGT-Funktionen einspeisen:

```
GGT ( 1 + ZZ 1000 ) ( 1 + ZZ 1000 )
ERGEBNIS: 6
```

Dieses Ergebnis könnte zum Beispiel dadurch zustande gekommen sein, daß der erste Aufruf von (1 + ZZ 1000) die Zahl 666 und der zweite Aufruf die Zahl 78 ergeben hat.

Aufgabe 3.63: Schreibe eine Simulationsprozedur, die am laufenden Band den GGT von zwei zufällig ausgewählten Zahlen berechnet, feststellt, ob die Zahlen teilerfremd sind oder nicht und die Anzahl der teilerfremden Paare zur Gesamtzahl der Versuche ins Verhältnis setzt. Vergleiche das Ergebnis des Simulationslaufes mit deinem Schätzwert (siehe Aufgabe 3.61).

Aufgabe 3.64: Die folgenden Zahlen sind ein Anfangsstück aus der Folge der **Fibonacci-Zahlen** (nach *Leonardo von Pisa*, genannt *Fibonacci*, ca. 1170 - 1240):

1 1 2 3 5 8 13 21 34 55 89 144 233 ...

(a) Formuliere ein Bildungsgesetz für die Fibonacci-Zahlen.
(b) Schreibe eine Logo-Funktion FIB :N, welche die :N-te Fibonacci-Zahl als Ausgabe hat.
(c) Untersuche die Teilbarkeitsverhältnisse bei den Fibonacci-Zahlen. Hier ein Beispiel: In der obigen Folge scheinen aufeinanderfolgende Fibonacci-Zahlen teilerfremd zu sein. Ist das allgemein so?
(d) Spiele den Euklidischen Algorithmus in der Subtraktionsform (von Hand und "per Programm" bei eingeschaltetem Protokollmodus) mit benachbarten Fibonacci-Zahlen durch. Was stellst du fest?

Aufgabe 3.65: Einige der in diesem Abschnitt besprochenen Prozeduren liegen in mehreren Versionen vor. Entscheide dich jeweils für eine lauffähige "Arbeitsversion", gib ihr einen prägnanten Namen und füge sie den bisherigen Prozeduren der Teilbarkeits-Bibliothek hinzu. (Du kannst dabei entsprechend vorgehen wie in Aufgabe 3.46).

3.5 Das kleinste gemeinsame Vielfache

Der größte gemeinsame Teiler (ggT) zweier Zahlen hängt eng mit ihrem *kleinsten gemeinsamen Vielfachen* **(kgV)** zusammen. Zur Bestimmung der Vielfachen können wir die Prozeduren aus Abschnitt 3.2 verwenden.

Die Vielfachen der Zahl 4 sind zum Beispiel:

4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 ...

Die Vielfachen der Zahl 6 sind

6 12 18 24 30 36 42 48 54 60 66 72 78 84 90 96 102 108 ...

Gemeinsame Vielfache der Zahlen 4 und 6 sind

12 24 36 48 60 72 ...

Und das kleinste gemeinsame Vielfache von 4 und 6 ist schließlich die Zahl 12.

Aufgabe 3.66: Am obigen Beispiel scheinen alle gemeinsamen Vielfachen von 4 und 6 zugleich auch Vielfache von 12, also des kleinsten gemeinsamen Vielfachen, zu sein. Überprüfe dies an weiteren Beispielen.

Während jede Zahl nur endlich viele Teiler hat, hat sie unendlich viele Vielfache. Kann es in ungünstigen Fällen passieren, daß wir "unendlich" lange suchen müssen, um ein gemeinsames Vielfaches zweier Zahlen a und b zu finden? Nein! Denn auf jeden Fall ist das Produkt a * b dieser Zahlen ein gemeinsames Vielfaches. Das Produkt a * b ist sicher ein Vielfaches (nämlich das b-fache) der Zahl a und ebenfalls ein Vielfaches (nämlich das a-fache) der Zahl b. Unter Umständen ist es aber nicht das kleinste unter den Vielfachen, wie zum Beispiel im Falle a=4 und b=6, wo das Produkt 4 * 6 gleich 24, das kleinste gemeinsame Vielfache aber gleich 12 ist. Auf jeden Fall können wir beruhigt sein: um das kleinste gemeinsame Vielfache zweier Zahlen a und b zu finden, müssen wir schlimmstenfalls alle Zahlen zwischen 1 und a * b durchprobieren.

Die in der folgenden Aufgabe genannten Prozeduren kennen wir in ähnlicher Form schon aus dem vorigen Abschnitt über den größten gemeinsamen Teiler. Meist muß man dabei nur schon bekannte Funktionen miteinander kombinieren.

Aufgabe 3.67: Schreibe die folgenden Funktionen

    GEMEINSAME.VIELFACHE :A :B :OBERGRENZE
    KLEINSTES.ELEMENT :L
    KLEINSTES.GEMEINSAMES.VIELFACHES :A :B

Hinweis: Du kannst alle bisher erarbeiteten Funktionen benutzen; zum Beispiel auch die Funktion

    VIELFACHE :A :OBERGRENZE

aus Abschnitt 3.2.

Aufgabe 3.68: Ähnlich wie beim größten gemeinsamen Teiler kann man auch das kleinste gemeinsame Vielfache zweier Zahlen a und b über ihre Primfaktorzelegung ermitteln. Dies sei am folgenden Beispiel erläutert:

```
a   =  14700  =  2 * 2 * 3 * 5 * 5 * 7 * 7
b   =   3234  =  2 *     3 *             7 * 7 * 11
-------------------------------------------------------------
kgV(a , b)    =  2 * 2 * 3 * 5 * 5 * 7 * 7 * 11   =   161700
```

Schreibe eine Prozedur KGV.PFZ, mit der das kleinste gemeinsame Vielfache zweier Zahlen über diese Primfaktor-Methode ermittelt wird.

Aufgabe 3.69: Überprüfe die folgende Tabelle mit den bisher erarbeiteten KGV-Funktionen und baue sie aus.

```
  a  !  b  !  ggT  !  kgV  !  ggT * kgV  !  a * b
---------------------------------------------------
  2  !  3  !   1   !   6   !      6      !    6
  4  !  6  !   2   !  12   !     24      !   24
 15  ! 18  !   3   !  90   !    270      !  270
 88  ! 24  !   8   ! 264   !   2112      ! 2112
 ...

Tabelle 3.6
```

Betrachte weitere Beispiele. Stelle eine Vermutung auf.

Merke: Für den größten gemeinsamen Teiler und das kleinste gemeinsame Vielfache zweier natürlichen Zahlen a und b gilt die folgende Gesetzmäßigkeit:

ggT(a , b) * kgV(a , b) = a * b

Eine solche allgemeine Aussage kann natürlich nicht durch die Betrachtung von Beispielen bewiesen werden, auch wenn es noch so viele sind.

Aufgabe 3.70: (Schwere Aufgabe für Liebhaber!) Versuche, eine allgemeingültige Begründung für das obige Gesetz zu geben. Hinweis: du kannst z.B. die Primfaktorzerlegung der Zahlen verwenden.

Mit Hilfe des obigen Gesetzes können wir jetzt die KGV-Funktion auf die GGT-Funktion zurückführen:

```
PR KGV  :A  :B
 RUECKGABE  DIV  (:A * :B )  ( GGT  :A  :B )
ENDE
```

Noch einfacher geht es wohl kaum. Du siehst, es lohnt sich, gelegentlich etwas Mathematik zu treiben.

Aufgabe 3.71: Vergleiche die Laufzeiten der Funktionen KGV und KLEINSTES.GEMEINSAMES.VIELFACHES. Lege eine Tabelle an.

Aufgabe 3.72: Beim Radrennen auf einer Radrennbahn fahren die Rennfahrer näherungsweise mit konstanter Geschwindigkeit. Fahrer A benötigt pro Runde 35 Sekunden; Fahrer B 40 Sekunden. Sie starten zugleich auf der Start-und-Ziel-Linie. Nach welchen Zeiten treffen sie sich jeweils wieder auf der Start-und Ziel-Linie?

In Büchern zur Logo-Graphik findet man häufig Prozeduren der folgenden Art:

```
PR POLY  :S  :W
 VORWAERTS  :S
 RECHTS  :W
 POLY  :S  :W
ENDE
```

Hier einige Beispiele:

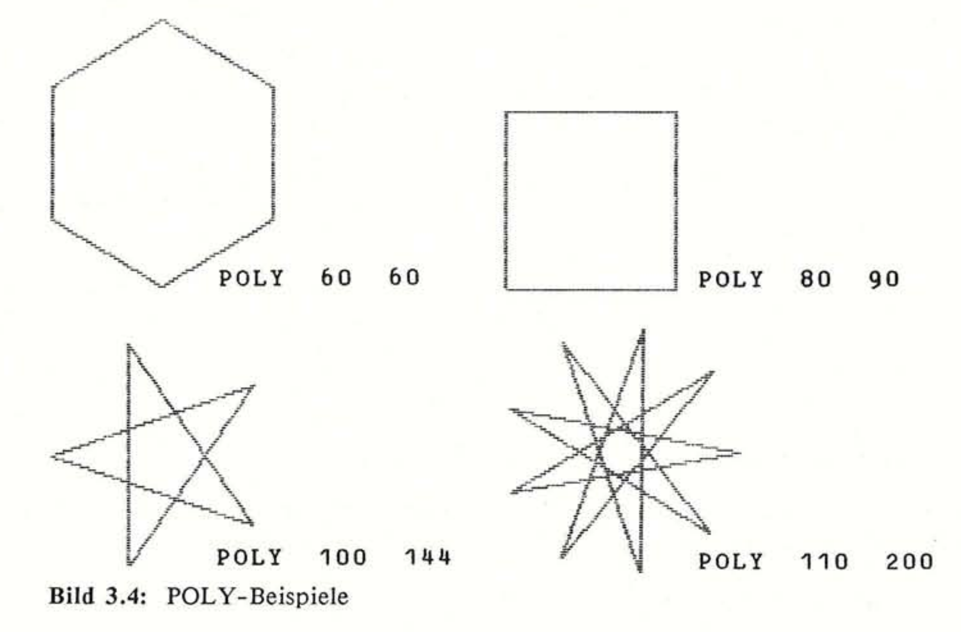


Bild 3.4: POLY-Beispiele

Die Prozedur POLY ist endrekursiv. Sie enthält keine STOP-Bedingung und läuft endlos weiter. Im Laufe der Zeit überfährt der Igel nur noch Strecken, die er vorher schon längst einmal gezeichnet hat. Wie kann man wissen, wann eine POLY-Figur fertig ist? Dieser Frage wollen wir in den folgenden Aufgaben nachgehen.

Aufgabe 3.73: Beim Lauf von Poly entstehen Dreiecke, Vierecke, Fünfecke, ... und Stern-Polygone (*Polygon* = Vieleck) mit verschiedener "Strahlenzahl". Wie hängt die Eckenzahl von den Eingaben :W und :S ab? Begründe, daß sie überhaupt nicht von :S, sondern nur von :W abhängt. Vervollständige die folgende Tabelle:

```
Eingabe-Winkel    !   entstandene Figur
 :W  (in Grad)    !
---------------------------------------
        60        !    Sechseck
        72        !    Fünfeck
        90        !    Viereck
       120        !    Dreieck
       ...        !    ...

       144        !    5-strahliger Stern
       160        !    9-strahliger Stern
       200        !    9-strahliger Stern
       220        !   18-strahliger Stern
       ...        !    ...
```

Tabelle 3.7

Aufgabe 3.74: Baue in die Tabelle eine dritte Spalte mit der Überschrift "Winkel * Eckenzahl" ein und fülle sie aus. Was stellst du fest?

Aufgabe 3.75: Schreibe zu POLY eine zweite Prozedur ZOLY, bei der der Igel die Strecke :S nach jedem Vorwärts-Lauf sofort wieder zurück läuft.

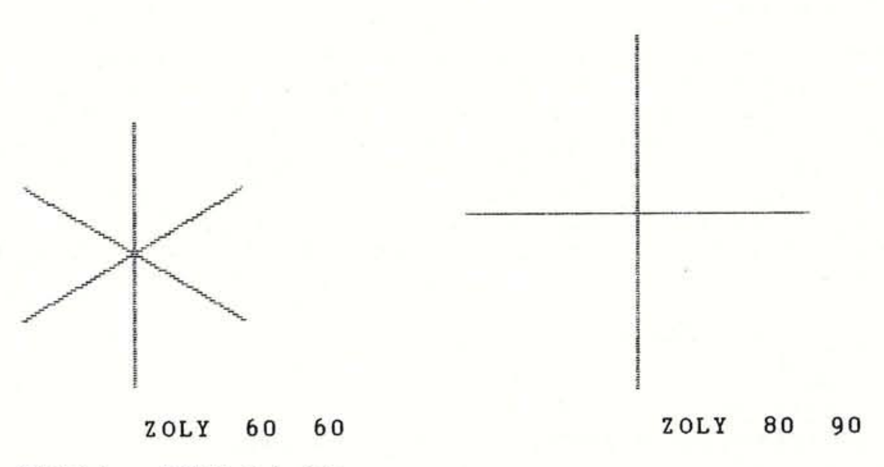


Bild 3.5 a: ZOLY-Beispiele

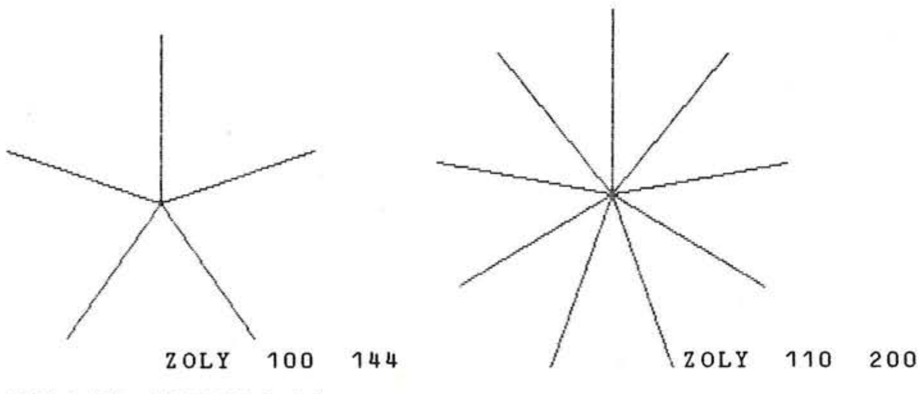


Bild 3.5 b: ZOLY-Beispiele

Aufgabe 3.76: Schreibe eine Prozedur DOLY, bei der die POLY- und die ZOLY-Figuren zeitlich parallel entstehen.

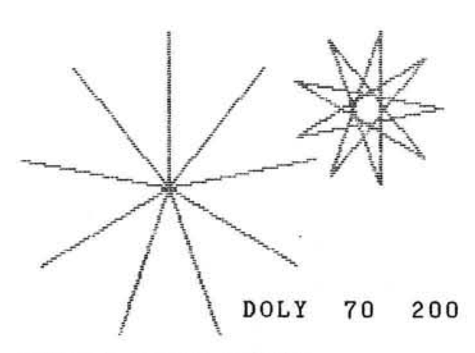


Bild 3.6: DOLY-Beispiel

Wenn man in der vorigen Aufgabe nur mit einem Igel arbeitet, dann muß man ihn immer zwischen der POLY- und der ZOLY-Teilfigur hin- und her springen lassen. Damit man weiß, wo man das nächste Mal weitermachen muß, muß man sich vor dem Sprung jeweils die alte Position und Blickrichtung in der POLY- bzw. ZOLY-Figur merken. Wenn man zwei Igel hätte, die unabhängig voneinander arbeiten, ginge das einfacher. Man würde einfach einen Igel für die POLY- und einen zweiten Igel für die ZOLY-Figur nehmen. Solche mehrfachen Igel gibt es im Commodore-Logo in Form von **sprites** (deutsch: **Feen**). Hier eine Sprite-Lösung unserer DOLY-Prozedur.

```
PR DOLY.SPRITE  :S  :W
 BILD
 AN  0
 AUFKURS  0
 STIFTHOCH  ( AUFXY  50  0 )  STIFTAB
 AN  1
 AUFKURS  0
 STIFTHOCH  ( AUFXY  (-50)  0 )  STIFTAB
 DOLY.SPRITE.1  :S  :W
ENDE
```

```
PR DOLY.SPRITE.1  :S  :W
 AN  0
 VORWAERTS  :S
 RECHTS  :W
 AN  1
 VORWAERTS  :S
 RUECKWAERTS  :S
 RECHTS  :W
 DOLY.SPRITE.1  :S  :W
ENDE
```

Erläuterungen zur Prozedur DOLY.SPRITE: Durch den Befehl BILD wird der Graphik-Bildschirm eingeschaltet und der Igel in die Mitte (Punkt 0;0) gesetzt. Durch den Befehl AN :N wird die Fee (sprite) Nr. :N aktiviert. Alle Igelbefehle werden bis zum nächsten AN-Befehl von dieser Fee ausgeführt. Der Wert von :N darf zwischen 0 und 7 liegen. Fee Nr. 0 ist der gewöhnliche Igel. Mit Hilfe der Routinen, die auf der Logo-Utilities-Diskette in der Datei SPRITES gespeichert sind, kann man den Feen verschiedene Formen geben. Im obigen Fall hat Fee Nr. 1 keine Form; sie ist unsichtbar und verhält sich ähnlich wie der gewöhnliche Igel, wenn man ihn mit Hilfe des Befehls VERSTECKIGEL (kurz VI) unsichtbar gemacht hat.

Der Befehl AUFKURS :R bewirkt, daß sich die angesprochene Fee in die Richtung :R dreht. Richtung 0 ist "Norden", Richtung 90 ist "Osten", Richtung 180 ist "Süden" und Richtung 270 ist "Westen". Auch ganzzahlige Zwischenwerte sind als *Windrosenwerte* möglich.

Der Befehl STIFTHOCH bewirkt, daß sich die Fee bewegt, ohne eine Spur hinter sich herzuziehen. Nach STIFTAB wird wieder eine Spur gezogen. Der Befehl AUFXY setzt die aktive Fee auf die angegebenen Koordinaten.

Die Hilfsprozedur DOLY.SPRITE.1 ist wieder endrekursiv.

Aufgabe 3.77: Rufe die Prozedur DOLY.SPRITE im Protokollmodus auf und beobachte genau, wie die Linien gezogen werden.

Aufgabe 3.78: Falls du mit Aufgabe 3.74 noch nicht ganz klar gekommen bist, dann versuche, sie jetzt zu beantworten.

Aufgabe 3.79: Wie müßte eine STOP-Bedingung für die POLY-Prozedur aussehen, damit jede Linie des Vielecks genau einmal überfahren wird?

Aufgabe 3.80: Entscheide dich, welche der Funktionen dieses Abschnitts du in die Teilbarkeits-Bibliothek aufnehmen willst. Bereinige die augenblickliche Arbeitsumgebung und füge die neuen Prozeduren zur Datei TEILBARKEIT hinzu. (Du kannst entsprechend vorgehen wie in Aufgabe 3.65).

Kapitel 4: Brüche und Anwendungen

4.1 Brüche

Ein Bruch besteht im wesentlichen aus zwei Dingen: einem Zähler und einem Nenner. Üblicherweise schreibt man einen Bruch etwa so:

$$\frac{3}{4}$$

Dabei ist die Zahl 3 der **Zähler** und die Zahl 4 der **Nenner**. Der Bruchstrich dient dazu, den Zähler deutlich vom Nenner zu trennen. In Logo ist es sinnvoll, den Bruch als eine Liste zu schreiben, die aus zwei Elementen besteht: Das erste Element ist der Zähler, das letzte der Nenner. Der obige Bruch sieht dann in Logo so aus:

```
[ 3  4 ]
```

Unser Ziel ist es, Logo die Anfangsgründe der Bruchrechnung beizubringen. Dazu gehört auch, daß man Brüche zerlegen und zusammensetzen kann. Hier einige grundlegende Hilfsfunktionen zur Bruchrechnung:

```
PR ZAEHLER  :BRUCH
 RUECKGABE  ERSTES  :BRUCH
ENDE

PR NENNER  :BRUCH
 RUECKGABE  LETZTES  :BRUCH
ENDE

PR BRUCH  :A  :B
 WENN  :B = 0  DANN
       DRUCKEZEILE [ NENNER NULL ! ] AUSSTIEG
 RUECKGABE ( LISTE  :A  :B )
ENDE
```

Hier einige Beispiele im Direktbetrieb:

```
ZAEHLER [ 5  7 ]
ERGEBNIS: 5

NENNER [ 19  128 ]
ERGEBNIS: 128

BRUCH 13 24
ERGEBNIS: [ 13  24 ]

BRUCH 5 0
NENNER NULL !

ZAEHLER BRUCH 4 9
ERGEBNIS: 4
```

Was die Funktionen ZAEHLER und NENNER machen, ist wohl klar. Die Funktion BRUCH fügt ihre beiden Eingabewerte zu einem Bruch (in Listendarstellung) zusammen.

Nun zum Rechnen mit Brüchen. Wenn man zwei Brüche addieren will, dann schreibt man das meist so:

$$\frac{2}{15} + \frac{12}{33} = \frac{82}{165}$$

In Logo können wir das Rechenzeichen nicht zwischen die Brüche setzen; wir müssen statt dessen eine Funktion schreiben, die als Eingabeparameter die beiden Brüche hat. Etwa so:

```
BRUCH.ADD  [ 2  15 ]  [ 12  33 ]
ERGEBNIS:  [ 82  165 ]
```

Der Funktionsname BRUCH.ADD steht dabei für "Bruchaddition". Die anderen Grundrechenarten für die Brüche wollen wir mit BRUCH.SUB, BRUCH.MULT und BRUCH.DIV bezeichnen.

```
PR BRUCH.ADD  :R  :S
 RUECKGABE  BRUCH ( ( ZAEHLER :R ) * ( NENNER :S ) + __
                    ( ZAEHLER :S ) * ( NENNER :R ) ) __
 _____________    ( NENNER :R ) * ( NENNER :S )
ENDE
```

Die Eingabeparameter :R und :S sollen Variable für Brüche darstellen; zum Beispiel :R = [2 15] und :S = [12 33].

```
BRUCH.ADD  [  2  15 ]   [ 12  33 ]
ERGEBNIS:  [ 246  495 ]
```

Das Ergebnis liegt noch nicht ganz in der gewünschten Form vor. Wir sollten noch kürzen. Hierfür ist die GGT-Funktion aus Kapitel 3, Abschnitt 3.4 sehr nützlich.

```
PR KUERZEN  :B
 LOKAL "Z
 SETZE "Z  ZAEHLER  :B
 LOKAL "N
 SETZE "N  NENNER :B
 LOKAL "D
 SETZE "D  GGT  ( ABS :Z )  ( ABS :N )
 RUECKGABE  BRUCH  ( DIV  :Z  :D )  ( DIV  :N  :D )
ENDE
```

```
KUERZEN   [ 246  495 ]
ERGEBNIS: [ 82  165 ]
```

Aufgabe 4.1: Schreibe eine Version von KUERZEN, bei der dafür gesorgt wird, daß der Nenner des Ergebnisses stets als positive Zahl erscheint. Zum Beispiel so:

```
KUERZEN     [  4  -6 ]
ERGEBNIS:   [ -2   3 ]
```

Aufgabe 4.2: Baue die Funktion KUERZEN so in die BRUCH.ADD-Funktion ein, daß das Ergebnis gleich in gekürzter Form ausgegeben wird.

Aufgabe 4.3: Selbst die verbesserte BRUCH.ADD-Version weist noch Schwachstellen auf. Die Zwischenergebnisse können unnötig groß werden. Schreibe eine Funktion HAUPTNENNER und baue sie so in die Funktion BRUCH.ADD ein, daß mit möglichst kleinen Zwischenergebnissen gerechnet wird.

Aufgabe 4.4: Schreibe Funktionen

NEGATIV :B
BRUCH.SUB :R :S
BRUCH.MULT :R :S
BRUCH.GROESSER? :R :S
BRUCH.KLEINER? :R :S
BRUCH.GLEICH? :R :S

Zur Division sollte man den **Kehrwert** benutzen:

```
PR KEHRWERT  :B
 RUECKGABE  BRUCH  ( NENNER :B )  ( ZAEHLER :B )
ENDE
```

```
KEHRWERT   [ 5  7 ]
ERGEBNIS:  [ 7  5 ]
```

Dann läßt sich die Funktion BRUCH.DIV so schreiben:

```
PR BRUCH.DIV  :R  :S
 RUECKGABE  BRUCH.MULT  :R  KEHRWERT  :S
ENDE
```

Aufgabe 4.5: Die Funktion KEHRWERT enthält noch eine Schwachstelle; welche wohl? Verbessere dies!

Logo beherrscht jetzt die Anfangsgründe der Bruchrechnung. Aber damit ist der Bereich der Bruchrechnung noch längst nicht erschöpft. Hier einige Übungen für fortgeschrittene Logo-Programmierer.

Aufgabe 4.6: Schreibe eine Version von KUERZEN, die völlig ohne die Hilfsvariablen Z, N und D auskommt.

Aufgabe 4.7: Baue die Funktionen zur Bruchrechnung so aus, daß Logo auch Doppel- und Mehrfachbrüche verarbeiten kann. Zum Beispiel so:

```
BRUCH.ADD  [ [ 2  3 ]  [ 4  5 ] ]  [ [ 6  7 ]  [ 8  9 ] ]
ERGEBNIS:  [ 151  84 ]
```

In der üblichen Schreibweise sieht die letzte Rechnung so aus:

$$\frac{\frac{2}{3}}{\frac{4}{5}} + \frac{\frac{6}{7}}{\frac{8}{9}} = \frac{151}{84}$$

Aufgabe 4.8: Baue die Bruchrechnung so aus, daß Logo auch Dezimalbrüche verarbeiten kann. Dazu gehört natürlich die Umwandlung von Dezimalbrüchen in gewöhnliche Brüche.

Aufgabe 4.9: Schreibe eine Prozedur DEZIMALBRUCH.AUSDRUCK zur Darstellung von gewöhnlichen Brüchen als Dezimalbrüche; zum Beispiel so:

```
DEZIMALBRUCH.AUSDRUCK [ 1 7 ]
0.142857142857142857 ...
```

Aufgabe 4.10: Schreibe eine Funktion DEZIMALBRUCH zur Umwandlung eines gewöhnlichen Bruches in einen Dezimalbruch; etwa so:

```
DEZIMALBRUCH [ 5 8 ]
ERGEBNIS: [ 0 , 6 2 5 ]
DEZIMALBRUCH [ 51 14 ]
ERGEBNIS: [ 3 , 6 PERIODE 4 2 8 5 7 1 ]
```

Untersuche, wann es zu periodischen Dezimalbrüchen kommt.

Aufgabe 4.11: (a) Schreibe eine Funktion EULER :N, welche die Anzahl der zwischen 1 und :N liegenden zu :N teilerfremden ganzen Zahlen ermittelt. (Du kannst natürlich alle notwendigen oder hilfreichen Funktionen aus Kapitel 3 benutzen). Hier einige Aufrufe von EULER:

```
EULER 10
ERGEBNIS: 4          (nämlich: 1, 3, 7 und 9)

EULER 7
ERGEBNIS: 6          (nämlich: 1, 2, 3, 4, 5 und 6 )
```

(b) Stelle eine Tabelle der folgenden Art auf:

Bruch	! Nenner (:N)	! EULER :N	! Periodenlänge
[1 3]	! 3	! 2	! 1
[1 6]	! 6	! 2	! 1
[1 7]	! 7	! 6	! 6
[1 9]	! 9	! 6	! 1
[1 11]	! 11	! 10	! 2
[1 13]	! 13	! 12	! 6

Tabelle 4.1

Betrachte weitere Beispiele. Untersuche, wie die Periodenlänge mit dem Wert der EULER-Funktion zusammenhängt.

Aufgabe 4.12: Erweitere die Bruchrechen-Funktionen so, daß Logo auch mit "gemischten Brüchen" rechnen kann; also mit Brüchen der Form 2 1/4 (= 2 + 1/4 = 9/4) usw.

4.2 Mischungsrechnung

Sabine hat eine Flasche Sirup gekauft. Durch Verdünnen mit Wasser kann sie ihr Lieblingsgetränk herstellen. Damit die Mischung gut schmeckt, muß sie etwa einen Teil Sirup mit fünf Teilen Wasser mischen. Die *Konzentration* des Sirups ist dann "eins zu sechs", also 1/6 oder etwa 16.7 %. Die folgende Funktion gibt jeweils als Funktionswert die Konzentration des Sirups zurück:

```
PR KONZENTRATION  :TEILE.SIRUP  :TEILE.WASSER
 RUECKGABE  :TEILE.SIRUP / (:TEILE.SIRUP + :TEILE.WASSER)
ENDE
```

Einige Beispiele:

KONZENTRATION 1 4
ERGEBNIS: 0.2 (entspricht 20 %)

KONZENTRATION 3 8
ERGEBNIS: 0.272727

KONZENTRATION 1 0
ERGEBNIS: 1 (entspricht 100 %)

Für eine Geburtstagsfeier will Sabine 6 Liter des Getränks mit einer Konzentration von 15 % herstellen. Wie muß sie den Sirup mit dem Wasser mischen?

Aufgabe 4.13: (a) Spiele etwas mit der Funktion KONZENTRATION und versuche dadurch, das geeignete Mischungsverhältnis herauszufinden.
(b) Berechne aus dem Mischungsverhältnis die Sirup- und die Wassermenge, die notwendig sind, um 6 Liter Mischung herzustellen.

Manchmal kommt man durch etwas Nachdenken eher zum Ziel. In einer 15-prozentigen Lösung kommen doch offenbar 15 Teile Sirup auf 85 Teile Wasser. Das Mischungsverhältnis muß also 15 : 85 oder 3 : 17 sein. Damit besteht die Mischung aus (3/20) * 6 Litern Sirup und (17/20) * 6 Litern Wasser, also aus 0.9 Litern Sirup und 5.1 Litern Wasser.

Sabine wollte aber gleich mit den Flüssigkeiten selbst experimentieren und war dabei etwas voreilig. Jetzt steht sie mit je einem 5-Liter-Gefäß aus 12-prozentiger und 26-prozentiger Mischung da. Wie müßte sie diese beiden Konzentrationen zusammenmischen, um eine 15-prozentige Konzentration zu bekommen?

Wenn sie die beiden Mischungen zu gleichen Anteilen vermischen würde, müßte der Grad der neuen Konzentration in der Mitte zwischen den alten Konzentrationen, also bei 19 Prozent liegen. Der Anteil der 12-prozentigen Mischung muß also erhöht und der Anteil der 26-prozentigen Mischung verringert werden.

Nehmen wir einmal an, Sabine mischt a Teile der 12-prozentigen Mischung mit b Teilen der 26-prozentigen Mischung zusammen. In den a Teilen der 12-prozentigen Mischung sind

$$a * \frac{12}{100} \text{ Teile Sirup}$$

und in den b Teilen 26-prozentiger Mischung sind

$$b * \frac{26}{100} \quad \text{Teile Sirup}$$

enthalten. Insgesamt besteht die neue Mischung also aus

$$a * \frac{12}{100} + b * \frac{26}{100} \quad \text{Teilen Sirup.}$$

Da die neue Gesamtmischung aus (a + b) Teilen Flüssigkeit besteht, beträgt die Konzentration der neuen Mischung:

$$k = \frac{a * \frac{12}{100} + b * \frac{26}{100}}{a + b} \qquad (*)$$

Aufgabe 4.14: Berechne aus dieser Gleichung für k=15/100 und a+b=6 die Werte für a und b.

Manchmal werden solche Rechnungen durchsichtiger, wenn man sie mit Variablen an Stelle von festen Zahlenwerten durchführt. Wir wollen also einmal an Stelle der im obigen Beispiel gegebenen Konzentrationen folgende Variable verwenden:

```
k1  an Stelle von 12 %
k2  an Stelle von 26 %
k   an Stelle der "Zielkonzentration" von 15 %
```

Dann lautet die letzte Gleichung

$$k = \frac{a * k1 + b * k2}{a + b} \qquad (**)$$

Hieraus folgt sofort

$$(a + b) * k = a * k1 + b * k2$$

beziehungsweise

$$a * (k - k1) = b * (k2 - k)$$

und schließlich:

$$\frac{a}{b} = \frac{(k2 - k)}{(k - k1)} \qquad (***)$$

Mit Hilfe von dieser Gleichung können wir nun leicht eine Logo-Funktion schreiben, die uns das gesuchte Mischungsverhältnis als Funktionswert zurückgibt:

```
PR MISCHUNGSVERHAELTNIS  :K1  :K2  :K
 RUECKGABE  (:K2 - :K) / (:K - :K1)
ENDE
```

Ein Beispiel:

MISCHUNGSVERHAELTNIS 12 26 15
ERGEBNIS: 3.66666

Das heißt, wir müssen 3.66666 Teile der 12-prozentigen mit einem Teil der 26-prozentigen Lösung mischen. Probe (auch im Logo-Direktbetrieb möglich):

$$(3.66666 * 12/100 + 1 * 26/100) / 4.66666 = 15/100$$

Bei manchen Anwendungen kommen die Konzentrationen nur als ganzzahlige Prozentzahlen vor, und man möchte das Mischungsverhältnis auch als echten Bruch haben. Mit unseren Kenntnissen aus der Logo-Bruchrechnung ist dies ebenfalls ganz einfach zu erreichen.

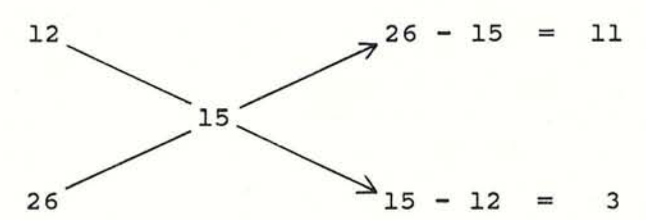
```
PR MISCHUNGSVERHAELTNIS.BRUCH  :K1  :K2  :K
 RUECKGABE  KUERZEN  BRUCH  (:K2 - :K)  (:K - :K1)
ENDE
```

Beispiel:

MISCHUNGSVERHAELTNIS.BRUCH 12 26 15
ERGEBNIS: [11 3]

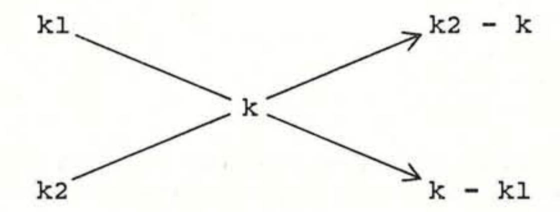
Probe: 11 / 3 = 3.66666

Der in der Formel (***) steckende Sachverhalt wird manchmal in der Form des **Mischungskreuzes** dargestellt. Hier zunächst ein Beispiel mit den festen Zahlen von oben:



Ergebnis: Um die Ausgangslösungen von 12-prozentiger und 26-prozentiger Konzentration zu einer 15-prozentigen Lösung zusammenzumischen, benötigt man 11 Teile der 12-prozentigen und 3 Teile der 26-prozentigen Lösung.

Hier das allgemeine Schema (dabei sei k2 > k1):



Ergebnis: Um die Ausgangslösungen der Konzentration k1 und k2 (in Prozent) zu einer k-prozentigen Lösung zusammenzumischen, benötigt man (k2 - k) Teile der Konzentration k1 und (k - k1) Teile der Konzentration k2.

Aufgabe 4.15: Schreibe eine Funktion

MISCHUNGSMENGEN :K1 :K2 :K :GESAMTMENGE ,

die als Zweierliste die beiden zusammenzumischenden Mengen zurückgibt; zum Beispiel:

MISCHUNGSMENGEN 12 26 15 6
ERGEBNIS: [4.71428 1.28571]   (Beachte den Rundungsfehler).

Aufgabe 4.16: Eine Küchenspüle hat zwei Zuleitungen. Aus der einen kommt heißes Wasser von 60 Grad und aus der anderen kaltes Wasser von 15 Grad. In welchem Verhältnis muß man die Wassermengen mischen, um Spülwasser von 30 Grad zu bekommen?

Aufgabe 4.17: Wieviel destilliertes Wasser muß man zu 300 ml 90-prozentigem Äthylalkohol hinzugießen, damit man eine 40-prozentige Lösung erhält?

Aufgabe 4.18: Aus 60 kg Messing mit 72 % Kupfergehalt soll durch Hinzuschmelzen von Messing mit 56 % Kupfergehalt Messing mit 60 % Kupfergehalt hergestellt werden.

4.3 Zinseszinsen und geometrisches Wachstum

Ein Hinweis zum Begriff des Prozentsatzes: Bei einem Zinssatz von 4.5 % betragen die Jahreszinsen eines Kapitals von 200 DM:

```
    4.5 % von 200 DM
 =  200 * 4.5 * 1/100 DM
 =  200 * 4.5 / 100 DM
 =  200 * 0.045 DM
 =  9 DM
```

Die Sprechweise "4.5 % von etwas" bedeutet also dasselbe wie "4.5 Hundertstel von etwas" oder "das 0.045-fache von etwas".

In Programmen zur Zinsrechnung taucht als Eingabewert oft ein Zinssatz auf (wie zum Beispiel oben in der Funktion NEUER.KONTOSTAND). Logo erwartet, daß dieser Parameter durch eine Zahleneingabe bestückt wird; also nicht etwa durch eine Eingabe der Art "4.5%" oder "4.5 PROZENT". Das heißt, der Aufruf darf *nicht* lauten:

NEUER.KONTOSTAND 200 4.5%
bzw.
NEUER.KONTOSTAND 200 4.5 PROZENT

Was ist aber die richtige Zahl, die wir eingeben müssen, um einen Prozentsatz von 4.5 Prozent zu beschreiben? Nun, nach dem oben Gesagten eigentlich 0.045. Doch dies ist etwas ungewohnt. Die Eingabe 4.5 erscheint "natürlicher" als die Eingabe 0.045. Wir können uns damit aus der Klemme helfen, daß wir vereinbaren:

Jegliche Eingabe von Prozentsätzen erfolgt in Prozent, also in Hundertsteln.

Dann müssen wir tatsächlich 4.5 eingeben, denn 4.5 Hundertstel sind ja gerade 4.5 Prozent. Wir wollen diese Vereinbarung durchweg einhalten.

Nimm einmal an, einer deiner Vorfahren hätte zu Christi Geburt auf deinen Namen einen Pfennig bei einem Zinssatz von 1 % angelegt. Hättest du gedacht, daß dieser Einsatz durch Zinseszinsen im Jahre 1985 auf etwa 3 782 400 DM angewachsen wäre, wenn ... ?

Aufgabe 4.19: Schreibe auf, was oben nach "wenn ... " alles stehen müßte.

Aufgabe 4.20: Anke eröffnet ein Sparbuch mit 200 DM zum Zinssatz von 4.5 %. Sie hebt nichts ab.
(a) Gib den Kontostand nach einem Jahr an.
(b) Gib den Kontostand nach Ablauf des zweiten, dritten, vierten Jahres an.
(c) Überprüfe die folgende Rechnung. Dabei seien K1, K2, K3 und K4 die Kontostände nach einem, zwei, drei und vier Jahren (in DM).

```
K1 = 200 * 1.045
K2 = K1  * 1.045
K3 = K2  * 1.045
K4 = K3  * 1.045
```

Begründe, warum man so rechnen darf.

Die folgende Funktion berechnet jeweils den Kontostand nach Ablauf eines Jahres:

```
PR NEUER.KONTOSTAND  :ALT  :ZINSSATZ
 RUECKGABE  :ALT * ( 1 + :ZINSSATZ / 100 )
ENDE
```

NEUER.KONTOSTAND 200 4.5
ERGEBNIS: 209

NEUER.KONTOSTAND 209 4.5
ERGEBNIS: 218.405

NEUER.KONTOSTAND (NEUER.KONTOSTAND 200 4.5) 4.5
ERGEBNIS: 218.405

Aufgabe 4.21: Verfolge den letzten Aufruf im Protokoll-Modus. Vergleiche ihn mit dem vorletzten Aufruf. Erkläre im einzelnen, wie Logo diesen Aufruf abarbeitet.

Mit dem folgenden Programm können wir den Verlauf der Zinseszinsberechnung über einen längeren Zeitraum verfolgen:

```
PR ZINSESZINSTABELLE  :ANFANGSKAPITAL :ZS  :LAUFZEIT
 ZINSESZINSTABELLE.HP  :ANFANGSKAPITAL  :ZS  :LAUFZEIT  0
ENDE

PR ZINSESZINSTABELLE.HP  :KAPITAL  :ZS  :L  :I
 WENN  :I > :L  DANN  RUECKKEHR
 ( DRUCKEZEILE  :I  :KAPITAL )
 ZINSESZINSTABELLE.HP (:K * (1 + :ZS/100)) :ZS :L (:I+1)
ENDE
```

Ein Beispiel:

```
ZINSESZINSTABELLE  200  4.5  16

 0   200
 1   209
 2   218.404
 3   228.233
 4   238.503
 5   249.236
 6   260.451
 7   272.171
 8   284.419
 9   297.218
10   310.593
11   324.569
12   339.175
13   354.437
14   370.387
15   387.054
16   404.472
```

Ankes Anfangskapitel hat sich also nach etwa 16 Jahren verdoppelt.

Aufgabe 4.22: Ermittle durch Experimente die Zeiten, nach denen sich das Anfangskapital in den folgenden Situationen verdoppelt:

Anfangskapital (in DM)	Zinssatz (in %)
100	2
200	2
300	2
100	3
200	3
300	3
100	4
200	4
300	4

Tabelle 4.2

Aufgabe 4.23: Zeige, daß die Zeit, nach der sich ein Anfangskapital verdoppelt (die *Verdopplungszeit*) nur vom Zinssatz und nicht vom Anfangskapital selbst abhängt.

Aufgabe 4.24: Schreibe eine Funktion VERDOPPLUNGSZEIT :ZS, die zu jedem eingegebenen Zinssatz die Verdopplungszeit (in ganzen Jahren) ermittelt. Ein Aufrufbeispiel:

```
VERDOPPLUNGSZEIT 4.5
ERGEBNIS: 16
```

Aufgabe 4.25: (a) Schreibe eine Prozedur VERDOPPLUNGSTABELLE, mit deren Hilfe du eine Tabelle der folgenden Art herstellen kannst.

ZINSSATZ (IN %)	VERDOPPLUNGSZEIT (IN GANZEN JAHREN)
0.5	139
1	70
1.5	47
...	...
7.5	10

Tabelle 4.3: Tabelle der Verdopplungszeiten

(b) Statte VERDOPPLUNGSTABELLE mit den Eingabeparametern :ANFANG, :SCHRITTWEITE und :ENDE aus, die es erlauben eine derartige Tabelle in beliebigen Grenzen und mit beliebiger Schrittweite zu erstellen.
(c) Füge den Tabellen eine dritte Spalte mit der Überschrift "ZINSSATZ * VERDOPPLUNGSZEIT" hinzu und fülle diese Spalte (natürlich "per Programm") aus.
(d) Versuche, für "kleine" Zinssätze bis etwa zu 7.5 % eine näherungsweise gültige Faustregel über das Produkt aus Zinssatz und Verdopplungszeit zu formulieren.

Aufgabe 4.26: Auf welchen Betrag wäre der zu Christi Geburt eingezahlte Pfennig bei einem Zinssatz von 3 % (bzw. 5 %) angewachsen?

Geometrisches Wachstum

Es sei B irgendeine Größe (zum Beispiel eine Bevölkerungszahl), die sich im Laufe der Zeit verändern kann. Man gibt dann die verstrichenen Zeiteinheiten jeweils in einer Klammer an: Zum Zeitpunkt 0 (z. B. im Ausgangsjahr) hat B den Anfangswert B(0). Mit B(1), B(2), B(3), ... wird der Wert der Größe nach einer, zwei, drei, ... Zeiteinheiten (z.B. Jahren) bezeichnet. Allgemein gibt B(n) den Wert der Größe B nach Ablauf von n Zeiteinheiten an.

Wenn sich die Größe B in gleich langen Zeitspannen immer um den gleichen Faktor ändert, dann sagt man: **"Die Größe B wächst geometrisch"**. Falls dieser (Wachstums-) Faktor z.B. gleich q ist, dann bedeutet dies:

```
B(1)  =  q * B(0)
B(2)  =  q * B(1)
B(3)  =  q * B(2)
...
```

allgemein:

```
B(n)  =  q * B(n-1)                                    (*)
```

Wenn wir mit einem festen Anfangswert B(0) anfangen, können wir bequem die Werte der Folge B(n) "hochrechnen". Nehmen wir das Beispiel aus Aufgabe 4.20. In diesem Fall war $B(0) = 200$ und $q = 1.045$.

```
B(1)  =  q * B(0)  =  1.045 * 200  =  209
B(2)  =  q * B(1)  =  1.045 * 209  =  218.404
B(3)  =  q * B(2)  =  1.045 * 218.404  =  228.233
B(4)  =  q * B(3)  =  1.045 * 228.233  =  238.503
...
```

Wenn das Gesetz (*) gilt, dann nennt man die Folge der Werte B(0), B(1), B(2), ... , B(n), ... eine **geometrische Folge**.

Aufgabe 4.27: Schreibe eine Prozedur

GEOMETRISCHE.FOLGE.AUSDRUCK :ANFANGSWERT :Q ,

mit der du die Glieder der angegebenen geometrischen Folge der Reihe nach ausdrucken kannst.

Aufgabe 4.28: Schreibe eine Funktion

GEOMETRISCHE.FOLGE :ANFANGSWERT :Q :OBERGRENZE ,

die als Ausgabe eine Liste mit den Elementen der entsprechenden geometrischen Folge bis zur angegebenen Obergrenze hat.

Aufgabe 4.29: Schreibe eine Funktion

GEOMETRISCHE.REIHE.AUSDRUCK :ANFANGSWERT :Q ,

mit der du die aufsummierten Glieder der angegebenen geometrischen Folge ausdrucken kannst. Zum Beispiel so:

```
GEOMETRISCHE.REIHE.AUSDRUCK  1  0.5

LFD.      FOLGEN-      REIHEN-
NR.:      WERT         WERT
-----------------------------
 1        1            1
 2        0.5          1.5
 3        0.25         1.75
 4        0.125        1.875
 ...
```

Aufgabe 4.30: Schreibe eine Funktion

GEOMETRISCHE.REIHE :ANFANGSWERT :Q :OBERGRENZE ,

die als Ausgabe eine Liste mit den Elementen der entsprechenden geometrischen Reihe bis zur angegebenen Obergrenze hat.

Aufgabe 4.31: In der Zeitung kannst du häufig Meldungen der Art lesen, wie sie in (a) und (b) wiedergegeben sind. In beiden Zeitungsausschnitten werden bestimmte Wachstumsprozesse angesprochen. Versuche, die Zeitungsberichte mit den in diesem Abschnitt entwickelten Werkzeugen zu analysieren. Suche nach weiteren solchen Zeitungsmeldungen in der Tageszeitung deiner Eltern und analysiere die Berichte.

(a) Die Einwohnerzahl Chinas:

Die Einwohnerzahl Chinas scheint die Milliardengrenze überschritten zu haben, meldet der amerikanische Environmental Fund in seinen unlängst veröffentlichten „Weltbevölkerungsschätzungen 1978“. Demnach hatte das bevölkerungsreichste Land der Erde Mitte des Jahres rund 1 003 900 000 Einwohner; bei einer Wachstumsrate von 2,3 Prozent sollte es bis zum Jahr 2000 die 1,37-Milliarden-Grenze erreicht haben. Die derzeitige Erdbevölkerung wird auf 4,365 Milliarden geschätzt; bei einer Wachstumsrate von zwei Prozent würde dies 6,5 Milliarden Menschen zur Jahrtausendwende bedeuten — ein gegenüber älteren Schätzungen langsameres Wachstum. Als die am schnellsten wachsende Region gilt Mittelamerika mit 3,2 Prozent (Westeuropa: plus 0,3 Prozent; die Bundesrepublik: minus 0,2 Prozent). Die Bundesrepublik ist mit 61,3 Millionen Einwohnern in der Liste der bevölkerungsstärksten Länder auf Platz zwölf hinter Mexiko (65,8 Millionen) zurückgefallen; bis zum Jahr 2000 dürften weitere fünf bis acht Staaten vorrücken. Dann wird auch Indien, das jetzt 656 Millionen Menschen beherbergt, über der Milliardengrenze liegen. *GH*

(b) Rentner-Club und Stadt Köln streiten sich um Regenwürmer:

Rentner-Club und Stadt Köln streiten sich um Regenwürmer

Köln. (dpa) Es klingt eigentlich eher wie eine Karnevalsposse, doch für die Beteiligten ist die Geschichte kein Spiel und schon gar nicht zum Lachen. Um eine Million Würmer geht der erbitterte Streit zwischen dem »Rentner-Aktiv-Club« aus der Eifel und der Stadt Köln.

Der seit Wochen währende Kampf um die Würmer hat sich mittlerweile so zugespitzt, daß sich das Kölner Landgericht wohl auf einen »Wurm-Prozeß« einrichten muß. Vermutlicher Streitwert: 40 000 Mark.

Einst hatte sich die Kölner Verwaltung eine Million der von den Rentnern in der Eifel gehegten und gepflegten Tierchen mitsamt ihrer heimatlichen Erde ausgeliehen, zum Zwecke der Umwandlung von Gartenabfall in Kompost. Alles verlief erfolgreich. Doch als die Wurm-Liebhaber verabredungsgemäß ihre »Leihgabe« wieder abholen wollten, war diese ausgerückt. Das Dilemma begann.

Es folgten Mahnbriefe der Rentner, die infolge der enormen Vermehrung der Tierchen ihr Anrecht auf inzwischen eine Milliarde Kompostwürmer betonten. Von der Stadt kamen juristische Expertisen, die von einer »bestimmten Erdmenge« mit einer »nicht genau bestimmten Anzahl Würmer« ausgingen und darauf hinwiesen, daß die Eifel-Tiere sich ja wohl mit ihren Kölner Artgenossen verbunden hätten. Wie also wollten die Rentner ihre Schützlinge finden?

Schließlich bot die Stadt eine Fuhre Komposterde mit einer ungezählten Menge Würmer an. Jedoch den Rentnern ist der »Wurmbesatz« der Erde zuwenig. Ein Sprecher nach einem »Ortstermin« am Dienstag: »Solchen Mist habe ich zu Hause auch.« Überhaupt wollen sie jetzt keine Würmer mehr wiederhaben. Die einzige Alternative zu einem Gerichtsverfahren heißt nach Angaben des Club-Gründers mittlerweile 40 000 Mark Schadensersatz.

Aufgabe 4.32: Vom Tierheim Reutlingen wurde das folgende Informationsblatt "Machen Sie sich mitschuldig?" verteilt. Analysiere die in diesem Blatt gemachten Angaben. Baue das Wachstumsmodell aus.

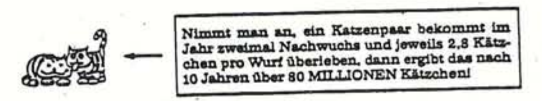
Machen Sie sich mitschuldig?

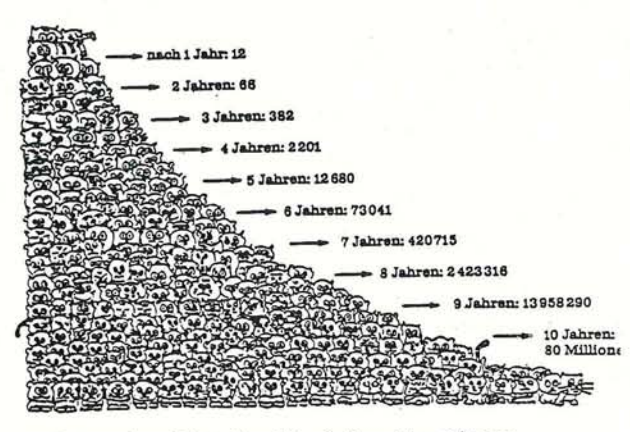
Millionen unerwünschter Katzen werden jedes Jahr geboren. Ende April bis September, der Hauptzeit für Katzengeburten, muß man herrenlose Katzen in Tierheimen sogar töten!

Weniger „glückliche" Tiere streunen umher. Ein Teil von ihnen wird überfahren, erschossen, mißhandelt. Viele landen als Versuchstiere in Labors. Die Überlebenden werden sehr schnell geschlechtsreif und gebären 5 oder 6 Kätzchen. So beginnt der Teufelskreis!

Jeder Katzenbesitzer, der sein Tier nicht sterilisieren läßt, macht sich mitschuldig.

Denken Sie daran: der Nachwuchs einer weiblichen Katze kann nach <u>10 Jahren über 80 Millionen Tiere betragen!!!</u>





Informationsblatt des Tierheims Reutlingen

Kapitel 5: Funktionen, Wertetafeln, Schaubilder

5.1 Wertetafeln

Jede Funktion stellt eine Zuordnung zwischen einem "Originalwert" (Urbild) und einem "Funktionswert" (Bild) her. Das Urbild wird meist als *Argument* bezeichnet. Es hat sich eingebürgert, daß man in der Mathematik für Funktionen meist die Symbole f, g, h, ... , für das Argument meist x und für den Funktionswert meist y verwendet. An Stelle der Aussage "dem Urbild x wird durch die Funktion f das Bild y zugeordnet" findet man auch die Sprechweisen "y ist der Funktionswert von f an der Stelle x" oder "x wird durch f auf y abgebildet"; in mathematischer Kurzschrift:

```
            f:  x  ——>  y      oder:
            y = f(x)

Beispiele:  y = f(x) = 3 * x + 5
            y = g(x) = x * x  -  5 * x  +  9
            y = h(x) = x * sin(x)
            ...
```

Es gibt hauptsächlich drei Methoden, die Zuordnung von Argumentwerten zu Funktionswerten darzustellen: die *Termdarstellung* (siehe oben), *Wertetafeln* und *Funktionsschaubilder*. Wertetafeln werden in diesem, Funktionsschaubilder im nächsten Abschnitt behandelt.

Eine **Wertetafel** ist eine Tabelle, die aus zwei Spalten besteht. In der linken Spalte stehen die Argumente und in der rechten Spalte die zugehörigen Funktionswerte. Ein kleiner Ausschnitt aus der Wertetafel der Funktion f mit

y = f(x) = x * x ("Quadratfunktion")

sieht zum Beispiel folgendermaßen aus:

```
   x     !  y
-----------------
   0     !  0
   0.5   !  0.25
   1     !  1
   1.5   !  2.25
   2     !  4
   2.5   !  6.25
   3     !  9
   3.5   ! 12.25
   4     ! 16
```

Tabelle 5.1

Wir wollen nun eine Prozedur schreiben, mit deren Hilfe wir solche Wertetafeln erstellen können. Die Prozedur soll so aufgerufen werden:

WERTETAFEL :FUNKTIONSNAME :ANFANG :ENDE :SCHRITT

Hier kommt eine bei anderen Programmiersprachen unbekannte Stärke von Logo zum Ausdruck: Wir können die Funktion selbst (in der obigen Form zum Beispiel über ihren Namen) als Eingabeparameter an die Prozedur WERTETAFEL übergeben. Bei praktisch allen anderen auf Mikrocomputern laufenden Programmiersprachen ist eine derartige Vorgehensweise unmöglich. Dort braucht man für jede Funktion eine eigene Wertetafel-Prozedur (wobei man meist das alte Wertetafel-Programm neu editiert, die alte Funktion herausnimmt und an ihre Stelle die neue Funktion in den Prozedurtext hineinschreibt). Daß Logo in diesem Punkte so viel mächtiger ist als die übrigen Sprachen, liegt an den Möglichkeiten der Listenverarbeitung, an der Unterscheidung zwischen dem Namen und dem Wert von Variablen (für die das "Quoten" ausschlaggebend ist) und am TUE-Befehl.

Der TUE-Befehl von Logo

Mit Hilfe des *TUE-Befehls* können wir Ausdrücke auswerten, die in Listen geschrieben sind. Durch Beispiele wird wohl am ehesten klar, was damit gemeint ist.

```
TUE [ 3 * 4 ]
ERGEBNIS: 12

TUE [ QW 6.25 ]
ERGEBNIS: 2.25

TUE [ SIN 30 ]
ERGEBNIS: 0.5

SETZE "X 17
TUE [ :X * :X ]
ERGEBNIS: 289

TUE [ 8 + :X ]
ERGEBNIS: 25

SETZE "F [ QW :X ]

WERT "F    ( bzw. :F )
ERGEBNIS: [ QW :X ]

ERSTES :F
ERGEBNIS: QW

LETZTES :F
ERGEBNIS: :X

WERT LETZTES :F
ERGEBNIS: 17

TUE WERT "F
ERGEBNIS: 4.12311

TUE :F
ERGEBNIS: 4.12311
```

QW 17
ERGEBNIS: 4.12311

Merke: Mit dem TUE-Befehl kann man die Auswertung von Listen erzwingen.

Wir wollen nun verschiedene Wertetafel-Prozeduren betrachten, die wir, wie üblich, durch Versionsnummern V1, V2, ... unterscheiden.

```
PR WERTETAFEL.V1 :FUNKTIONSNAME :ANFANG :ENDE :SCHRITT
 WERTETAFEL.V1.HP :FUNKTIONSNAME :ANFANG :ENDE :SCHRITT
ENDE

PR WERTETAFEL.V1.HP  :FN  :X  :ENDE  :SCHRITT
 WENN  :X > :ENDE  DANN  RUECKKEHR
 ( DRUCKEZEILE  :X  LEERSTELLEN  TUE WERT :FN )
 WERTETAFEL.V1.HP :FN ( :X + :SCHRITT ) :ENDE :SCHRITT
ENDE
```

LEERSTELLEN ist eine Hilfsfunktion, die ein aus drei Leerzeichen bestehendes Wort ausgibt:

```
PR LEERSTELLEN
 RUECKGABE ( WORT  ZEICHEN 32  ZEICHEN 32  ZEICHEN 32 )
ENDE
```

ZEICHEN ist eine Logo-Grundfunktion. Der Aufruf ZEICHEN :N gibt als Funktionswert das Zeichen zurück, dessen "ASCII-Wert" die Nummer :N hat. Die **ASCII-Werte** sind in einer Tabelle, der "ASCII-Tabelle", aufgeschrieben (ASCII steht für "American Standard Code for Information Interchange"; also: amerikanischer Standard-Code für den Informationsaustausch). Du findest sie in deinem Handbuch. Das **Leerzeichen** hat den ASCII-Wert 32. Die Logo-Grundfunktion WORT (siehe Kapitel 1, Abschnitt 1.7) fügt die drei Leerzeichen zu einem dreistelligen "Leerwort" zusammen.

Die Prozedur WERTETAFEL.V1 dient natürlich wieder nur dazu, das eigentliche (endrekursive) Arbeitspferd WERTETAFEL.V1.HP aufzurufen. Der Eingabeparameter :FN steht für "Funktionsname"; :FN enthält also nicht die Funktion selbst, sondern nur ihren *Namen*. Die Funktion erhalten wir in der Form einer Liste als WERT dieses Namens, also als WERT :FN. Durch den Aufruf TUE WERT :FN wird die Auswertung der Funktion an der augenblicklichen Stelle :X veranlaßt.

Im folgenden werden wir einige Beispiele betrachten. Vorher sollte am besten VERGISS NAMEN eingegeben werden, damit alle globalen Variablen gelöscht werden. Eine Bemerkung noch zum stellengerechten Ausdruck: Der besseren Lesbarkeit halber sind die nachfolgenden Wertetafeln in stellengerechter Darstellung gedruckt. Bei dir wird der Ausdruck nicht in der stellengerechten Form erscheinen. Bei den meisten Heimcomputern ist es ziemlich mühsam, stellengerechte Ausdrucke zu erreichen. Wenn du etwas geübter bist, kannst du dir Hilfsprozeduren für den stellengerechten Ausdruck schreiben. (Eine Gruppe solcher Prozeduren ist in dem Buch *"Programmieren lernen mit Logo"*, Hanser Verlag, beschrieben).

```
SETZE "F1 [ :X * :X * :X ]
WERTETAFEL.V1 "F1 0 6 0.5
0      0
0.5     0.125
1       1
1.5     3.375
2       8
2.5    15.625
3      27
3.5    42.875
4      64
4.5    91.125
5     125
5.5   166.375
6     216
```

Bei der Abarbeitung des letzten Aufrufs hat die Variable FN den Wert F1, und F1 hat den Wert [:X * :X * :X]. Man kann sich dies anhand des folgenden Bildes vorstellen:

FN ⟶ F1 ⟶ [:X * :X * :X]

```
SETZE "F2 [ 2 * :X * :X - 3 * QW :X ]
WERTETAFEL.V1 "F2 10 15 1
10   190.513
11   232.05
12   277.607
13   327.183
14   380.775
15   438.381
```

Aufgabe 5.1: Schreibe eine Funktion LEERSTELLEN :N, die ein Wort zurückgibt, das aus :N Leerstellen besteht.

Aufgabe 5.2: Was würde passieren, wenn du die Wertetafel-Funktion so aufrufst:
(a) WERTETAFEL.V1 F1 0 6 0.5
(b) WERTETAFEL.V1 :F2 10 15 1

Das letzte Beispiel entspricht dem Aufruf

WERTETAFEL.V1 [2 * :X * :X - 3 * QW :X] 10 15 1 (*)

Bei diesem Aufruf ist nicht der *Funktionsname*, sondern die *Funktionsliste* selbst eingegeben worden. Dies muß natürlich in der Prozedur WERTETAFEL.V1 an der Stelle zu Schwierigkeiten führen, wo diese Liste erst als der Wert der Variablen FUNKTIONSNAME ermittelt werden soll.

Andererseits können wir aber die obigen Prozeduren leicht so abändern, daß die Funktionsliste selbst (und nicht ihr Name) einzugeben ist. Dazu müssen wir nur den Befehl WERT vor :FN herausnehmen. Wir brauchen der Funktion dann nicht einmal einen Namen zu geben. Mit dieser kleinen Änderung würde der Aufruf in (*) ordentlich verarbeitet. Es gehört allerdings zum guten Programmierstil, den Eingabeparameter FUNKTIONSNAME umzubenennen. Denn jetzt wird ja eine Funktionsliste eingegeben.

```
PR WERTETAFEL.V2 :FUNKTIONSLISTE :ANFANG :ENDE :SCHRITT
 WERTETAFEL.V2.HP :FUNKTIONSLISTE :ANFANG :ENDE :SCHRITT
ENDE

PR WERTETAFEL.V2.HP  :FL  :X  :ENDE  :SCHRITT
 WENN  :X > :ENDE   DANN  RUECKKEHR
 ( DRUCKEZEILE  :X  LEERSTELLEN  TUE :FL )
 WERTETAFEL.V2.HP :FL ( :X + :SCHRITT ) :ENDE :SCHRITT
ENDE
```

Einige Beispiele:

```
WERTETAFEL.V2  [ 3 * :X  +  17 ]   (-5)  8  2
-5   2
-3   8
-1  14
 1  20
 3  26
 5  32
 7  38

WERTETAFEL.V2  [ 1.7 + 30 * :X - 4.905  * :X * :X ]  0  5
1
0     1.7
1    26.795
2    42.08
3    47.555
4    43.22
5    29.075
```

Der LIESLISTE-Befehl von Logo

Diese Stelle eignet sich gut, um einmal zu zeigen, wie man ein *Dialogprogramm* um die "Kernfunktionen" herum schreiben kann: eine typische Anwendungsmöglichkeit für den LIESLISTE-Befehl von Logo.

```
 1: PR WERTETAFEL.DIALOG
 2:  LOKAL "F   LOKAL "A   LOKAL "E   LOKAL "S
 3:  DRUCKEZEILE [  GIB EINEN FUNKTIONSTERM EIN:  ]
 4:  SETZE  "F  LIESLISTE
 5:  DRUCKEZEILE [  ANFANGSWERT: ]
 6:  SETZE  "A  ERSTES LIESLISTE
 7:  DRUCKEZEILE  [  ENDWERT:  ]
 8:  SETZE  "E  ERSTES LIESLISTE
 9:  DRUCKEZEILE  [ SCHRITTWEITE: ]
10:  SETZE  "S  ERSTES LIESLISTE
11:  WERTETAFEL.V1  :F  :A  :E  :S
12:  DRUCKEZEILE  [  NOCH EINE WERTETAFEL? (J/N):  ]
13:  WENN ERSTES LIESLISTE = "J  DANN WERTETAFEL.DIALOG
14: ENDE
```

Diese Prozedur wird im Anschluß an den folgenden Dialog erläutert:

```
Benutzer:   WERTETAFEL.DIALOG
Logo:       GIB EINEN FUNKTIONSTERM EIN:
Benutzer:   :X / ( 5  +  0.9 * :X  +  0.005 * :X * :X )
Logo:       ANFANGSWERT:
Benutzer:   0
Logo:       ENDWERT:
Benutzer:   120
Logo:       SCHRITTWEITE:
Benutzer:   10
Logo:
  0   0
 10   0.689655
 20   0.8
 30   0.821918
 40   0.816326
 50   0.8
 60   0.779221
 70   0.756757
 80   0.733945
 90   0.711462
100   0.689655
110   0.668693
120   0.648649
Logo: NOCH EINE WERTETAFEL? (J/N):
Benutzer: N
<Blinker>
```

Erläuterung der Prozedur WERTETAFEL.DIALOG: Die Variablen F, A, E und S stehen für Funktion, Anfangswert, Endwert und Schrittweite. Wenn Logo auf den Befehl LIESLISTE (kurz: LL) stößt, packt es alles, was bis zum nächsten Carriage-Return-Zeichen folgt, in eine Liste, die als Funktionswert von LIESLISTE zurückgegeben wird. Eingabe-Teile, die durch das Leerzeichen getrennt sind, geben verschiedenen Listenelemente ab. (Beim deutschen Logo für den APPLE II Computer ist an Stelle von LIESLISTE das Grundwort EINGABE zu verwenden).

Nach der Aufforderung "GIB EINEN FUNKTIONSTERM EIN:" wird durch den LIESLISTE-Befehl im obigen Dialog die Liste

[:X / (5 + 0.9 * :X + 0.005 * :X * :X)]

zurückgegeben und der lokalen Variablen F als Wert zugeordnet (Zeile 4). Dieser Funktionsname wird in Zeile 11 an die Prozedur WERTETAFEL.V1 weitergereicht. Nach der Aufforderung "ANFANGSWERT:" (Zeile 5) wird durch den LIESLISTE-Befehl die Liste [0] weitergegeben. Der Anfangswert ist das erste Element dieser Liste. Dem wird in dem Befehl

SETZE "A ERSTES LIESLISTE

Rechnung getragen (Zeile 6). In den Zeilen 7 und 8 geschieht Entsprechendes in bezug auf den Endwert und in den Zeilen 9 und 10 in bezug auf die Schrittweite.

Aufgabe 5.3: Ersetze im obigen Dialog den Befehl DRUCKEZEILE durch DRUCKE und beobachte, wie sich das auswirkt.

Die Prüfwörter von Logo (siehe Kapitel 2, Abschnitt 2.8) ermöglichen eine Wertetafel-Prozedur, bei der man nach Belieben entweder den Funktionsnamen oder die Funktionsliste eingeben kann.

```
PR WERTETAFEL  :F  :ANFANG  :ENDE  :SCHRITT
 WENN  NAME?  :F  DANN
       WERTETAFEL.V1  :F  :ANFANG  :ENDE  :SCHRITT
 WENN  LISTE? :F  DANN
       WERTETAFEL.V2  :F  :ANFANG  :ENDE  :SCHRITT
ENDE
```

Ein Beispiel:

```
SETZE  "F3   [  (SIN  :X) +  (COS :X) ]
WERTETAFEL  "F3  (-2)  3  0.5
-2      0.964491
-1.5    0.973443
-1      0.982395
-0.5    0.991197
 0      1
 0.5    1.00865
 1      1.0173
 1.5    1.02579
 2      1.03429
 2.5    1.04262
 3      1.05096
```

Dasselbe Ergebnis hätte man auch durch den Aufruf
WERTETAFEL [(SIN :X) + (COS :X)] (-2) 3 0.5
erhalten.

Aufgabe 5.4: Baue die Wertetafel-Prozeduren so aus, daß auch ein anderer Variablenname als X verwendet werden kann.

Aufgabe 5.5: Stelle die Wertetafel-Prozeduren zu einer Bibliothek (zum Beispiel unter dem Dateinamen WERTETAFELN) zusammen.

5.2 Funktionsschaubilder

Der Unterschied zwischen einer Wertetafel und einem Funktionsschaubild besteht im Prinzip nur darin, daß beim Funktionsschaubild nicht die Werte x und f(x) ausgedruckt werden, sondern daß stattdessen in einem (x/y)-Koordinatensystem ein Punkt an die Stelle (x / f(x)) gezeichnet wird.

Beim Zeichnen auf einem Computerbildschirm sind allerdings noch einige spezielle Probleme zu lösen. Sie ergeben sich aus der Begrenztheit des Bildschirms und aus der herstellerabhängigen Einteilung des Bildschirms in ein Zeilen- und Spaltenraster.

Die folgende Beschreibung bezieht sich zwar auf den Bildschirm des Commodore 64 Computers, sie ist aber leicht auf andere Computer mit Graphik-Bildschirmen zu übertragen.

Das deutsche Logo für den Commodore 64 Computer verfügt über zwei Graphikdarstellungen:

(1) Im **Vollbild-Modus** (englisch: *fullscreen mode*) wird der gesamte Bildschirm zur Darstellung von Graphiken verwendet. Die vier Ecken des Bildschirms haben dabei die folgenden Koordinaten:

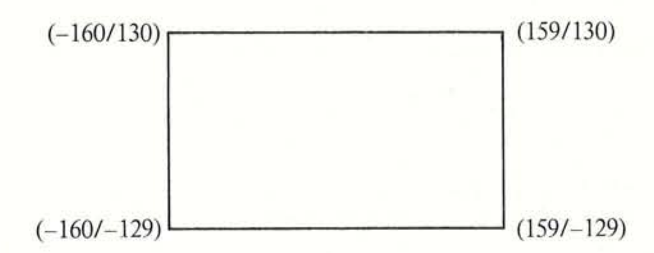


Bild 5.1: Vollbild-Modus

(2) Im **Teilbild-Modus** (englisch: *splitscreen mode*) wird der untere Bildschirmrand von vier Textzeilen überlagert.

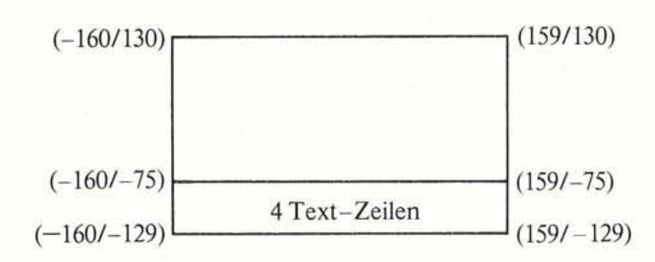


Bild 5.2: Teilbild-Modus

Im Teilbild-Modus ist also ein kleinerer Graphik-Bereich sichtbar als im Vollbild-Modus. Der von den vier Textzeilen verdeckte untere Bildschirmrand kann aber sowohl von Hand als auch vom Programm aus ein- oder ausgeblendet werden. Neben diesen beiden Graphik-Modi gibt es natürlich noch den gewöhnlichen Text-Modus, wo der gesamte Bildschirm für Textzeilen verwendet wird.

Die Umschaltung zwischen den einzelnen Modi erfolgt entweder mit Hilfe von Logo-Grundbefehlen oder mit Hilfe von Control-Tasten oder mit Hilfe der Funktionstasten:

Modus	Grundbefehl	Control-Taste	Funktions-Taste
Text	TEXTSCHIRM	Ctrl-T	F1
Teilbild	TEILBILD	Ctrl-S	F3
Vollbild	VOLLBILD	Ctrl-F	F5

Nach diesen technischen Vorbemerkungen nun zurück zu den Funktionsschaubildern. Bevor wir mit dem Zeichnen anfangen, müssen wir festlegen, welcher Ausschnitt der gewöhnlichen (x/y)-Ebene auf dem Bildschirm abgebildet werden soll und welchen Bildschirmausschnitt wir benutzen wollen. (Wir wollen das Problem so

lösen, daß wir jeden beliebigen Bildschirmausschnitt vorgeben können, in dem gezeichnet werden soll).

Um griffige Namen zur Verfügung zu haben, bezeichnen wir die Koordinaten der gewöhnlichen (x/y)-Ebene als die **Weltkoordinaten** und die Koordinaten auf dem Bildschirm als die **Bildschirmkoordinaten**.

Die Weltkoordinaten des Ausgangsrechtecks werden im folgenden mit :X und :Y bezeichnet und durch die "Randwerte" :XMIN, :XMAX, :YMIN und :YMAX begrenzt. Die Bildschirmkoordinaten werden mit :XB und :YB bezeichnet und durch die Randwerte :XBMIN, :XBMAX, :YBMIN und :YBMAX begrenzt. (Das "B" soll dabei an "Bildschirm" erinnern).

Die Wahl der Randwerte für die Weltkoordinaten ist völlig beliebig; die Bildschirm-Randwerte müssen dagegen den folgenden Einschränkungen genügen:

```
-160   ≤   :XBMIN   <   :XBMAX   ≤   159
-129   ≤   :YBMIN   <   :YBMAX   ≤   130
```

Ein typischer Satz von Eckwerten könnte zum Beispiel folgendermaßen aussehen:

```
(a) Weltkoordinaten:          :XMIN = -3        :XMAX = 8
                              :YMIN = -1.8      :YMAX = 5.5

(b) Bildschirmkoordinaten:    :XBMIN = -100     :XBMAX = 100
                              :YBMIN = -60      :YBMAX = 120
```

Wir stehen nun also vor dem Problem, die Punkte des gewählten Welt-Rechtecks auf die entsprechenden Punkte des Bildschirm-Rechtecks abzubilden.

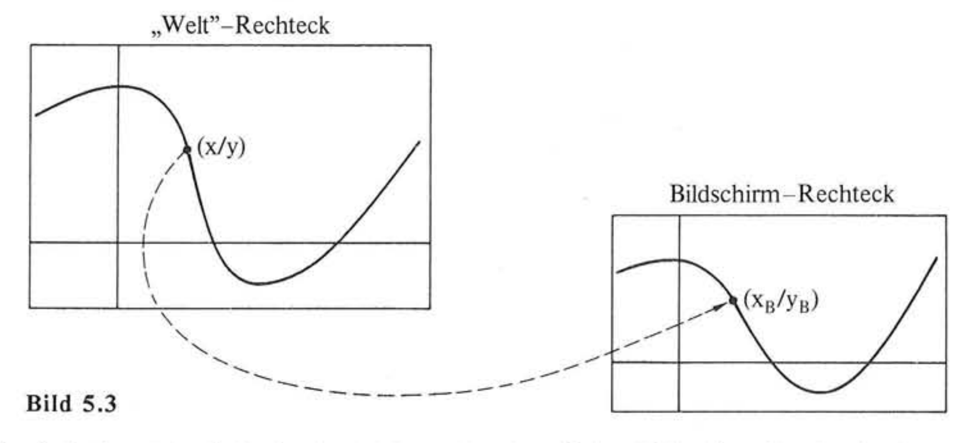


Bild 5.3

Die einfachste Möglichkeit, das Welt-Rechteck auf das Bildschirm-Rechteck abzubilden, besteht darin, die x-Werte und die y-Werte der Weltkoordinaten getrennt auf den Bildschirm zu übertragen.

Beginnen wir zunächst mit den x-Werten. Im Urbild können sie zwischen :XMIN und :XMAX variieren; ihre Bildwerte bewegen sich zwischen :XBMIN und :XBMAX. In anderen Worten ausgedrückt, liegt das Problem vor, das Intervall (:XMIN ; :XMAX) auf das Intervall (:XBMIN ; :XBMAX) abzubilden.

Die **Abbildung eines Intervalls auf ein anderes Intervall** ist ein Problem von allgemeinem Interesse. Deshalb wollen wir es hier in allgemeinem Zusammenhang und mit allgemeinen Bezeichnungen besprechen.

Wir wollen also das Intervall (a ; b) auf das Intervall (c ; d) abbilden. Die einfachste Möglichkeit hierfür liefert die zentrische Streckung. Das Bild des Punktes x aus (a ; b) wird im folgenden mit u bezeichnet.

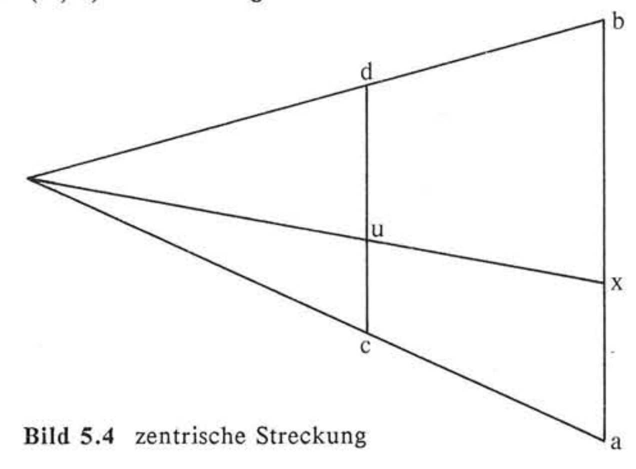


Bild 5.4 zentrische Streckung

Mit Hilfe des dritten Strahlensatzes erhalten wir die Gleichung:

$$\frac{u - c}{d - c} = \frac{x - a}{b - a}$$

Hieraus folgt:

$$u = (x - a) * \frac{d - c}{b - a} + c \qquad (*)$$

Die Transformationsfunktion, die uns zum gegebenen x-Wert den entsprechenden "Bildwert" u liefert, wollen wir mit TRANS bezeichnen. Wir können (*) deshalb auch so schreiben:

$$\text{TRANS}(x) = (x - a) * \frac{d - c}{b - a} + c \qquad (**)$$

```
Probe:  TRANS(a) = c
        TRANS(b) = d
```

Für unser Funktionsschaubild benötigen wir natürlich zwei solche Transformationsfunktionen: eine für die x-Werte und eine für die y-Werte der Weltkoordinaten. Wir nennen sie entsprechend TRANSX und TRANSY. Zunächst zu TRANSX. In diesem Fall ist a = :XMIN, b = :XMAX, c = :XBMIN und d = :XBMAX.

```
PR TRANSX :X
 RUECKGABE  ( :X - :XMIN ) *
 ____  ( :XBMAX - :XBMIN ) / ( :XMAX - :XMIN )  +  :XBMIN
ENDE
```

Die Transformationsfunktion für die y-Werte der Weltkoordinaten ergibt sich ganz entsprechend:

```
PR TRANSY :Y
 RUECKGABE  ( :Y - :YMIN ) *
 ____  ( :YBMAX - :YBMIN ) / ( :YMAX - :YMIN )  +  :YBMIN
ENDE
```

Nun wollen wir die Koordinatenachsen an die richtige Stelle auf den Bildschirm zeichnen. Um den Sachverhalt am Anfang nicht zu kompliziert zu machen, wollen wir im folgenden annehmen, daß der Koordinatenursprung im Inneren des Welt-Rechtecks liegt. Diese Einschränkung läßt sich später leicht aus dem Weg räumen. Die Prozedur zum Zeichnen der Koordinatenachsen soll einfach ACHSEN heißen. Sie hängt zweifellos von den acht Eck-Werten :XMIN, :XMAX, :YMIN, :YMAX, :XBMIN, :XBMAX, :YBMIN und :YBMAX ab. Wenn wir alle diese Werte als Eingabeparameter an die Prozedur ACHSEN übergeben, wird der Aufruf dieser Prozedur ziemlich umständlich. Wir stellen uns deshalb auf den Standpunkt, daß die Bildschirmkoordinaten konstant sind und legen sie ausnahmsweise als globale Variable fest. Etwa so:

```
PR MAXIMALER.RAHMEN!
 SETZE  "XBMIN  ( - 160 )
 SETZE  "XBMAX  159
 SETZE  "YBMIN  ( - 129 )
 SETZE  "YBMAX  130
ENDE
```

Eine Möglichkeit, den Rahmen im Dialog festzulegen, wäre zum Beispiel:

```
PR RAHMEN.DIALOG
 DR  [ XBMIN = ]   SETZE  "XBMIN   ERSTES LIESLISTE
 DR  [ XBMAX = ]   SETZE  "XBMAX   ERSTES LIESLISTE
 DR  [ YBMIN = ]   SETZE  "YBMIN   ERSTES LIESLISTE
 DR  [ YBMAX = ]   SETZE  "YBMAX   ERSTES LIESLISTE
ENDE
```

Falls wir unser späteres Schaubild einrahmen wollen, können wir dies mit Hilfe der folgenden Prozedur tun:

```
PR RAHMEN
 STIFTHOCH   AUFXY  :XBMIN  :YBMIN   STIFTAB
 AUFXY  :XBMIN  :YBMAX
 AUFXY  :XBMAX  :YBMAX
 AUFXY  :XBMAX  :YBMIN
 AUFXY  :XBMIN  :YBMIN
ENDE
```

Aufgabe 5.6: Informiere dich im Logo-Handbuch über die Befehle AUFX, AUFY und schreibe eine vereinfachte Prozedur RAHMEN mit diesen Befehlen.

Da die Werte für das Zeichenformat (Rahmen) als globale Variable festgelegt sind, benötigt die Prozedur ACHSEN als Eingabeparameter nur die Eckwerte der Weltkoordinaten :XMIN, :XMAX, :YMIN und :YMAX.

In welcher Höhe ist nun die x-Achse auf dem Bildschirm zu zeichnen? Alle Punkte der x-Achse haben in den Weltkoordinaten den y-Wert 0. Also müssen sie in den Bildschirmkoordinaten den Wert TRANSY 0 haben. Die x-Achse verläuft am Bildschirm also zwischen den Punkten

(:XBMIN / TRANSY 0) und (:XBMAX / TRANSY 0)

Entsprechend verläuft die y-Achse auf dem Bildschirm zwischen den Punkten

(TRANSX 0 / :YBMIN) und (TRANSX 0 / :YBMAX)

Damit sind wir nun in der Lage, die Achsen zu zeichnen.

```
PR ACHSEN  :XMIN  :XMAX  :YMIN  :YMAX
 VERSTECKIGEL
 LOKAL  "XB0  SETZE  "XB0  TRANSX 0
 LOKAL  "YB0  SETZE  "YB0  TRANSY 0
 WENN  AUSSERHALB?  :XB0  :YB0  DANN
     DRUCKEZEILE  [ URSPRUNG AUSSERHALB ! ]  AUSSTIEG
 WENN  :RAHMEN.OPTION  =  "WAHR  DANN  RAHMEN
 STIFTHOCH  AUFXY  :XBMIN  :YB0
 STIFTAB    AUFXY  :XBMAX  :YB0
 STIFTHOCH  AUFXY  :XB0  :YBMIN
 STIFTAB    AUFXY  :XB0  :YBMAX
ENDE
```

Erläuterung zur Prozedur ACHSEN: Wenn man mit der Koordinatengeometrie an Stelle der Igelgeometrie arbeitet, ist es besser, den Igel zu verstecken. Die Prüffunktion AUSSERHALB? in Zeile 5 wird weiter unten erläutert. Sie gibt WAHR ober FALSCH zurück, je nachdem, ob der Ursprung außerhalb des Zeichenfeldes liegt oder nicht. Wenn der Ursprung außerhalb liegt, wird eine Fehlermeldung gedruckt. Wir wollen noch die globale Variable RAHMEN.OPTION benutzen. Ist ihr Wert WAHR, so soll ein Rahmen gezeichnet werden, sonst nicht. Man kann dies z.B. im Dialogbetrieb steuern, indem man den Befehl SETZE "RAHMEN.OPTION "WAHR beziehungsweise SETZE "RAMEN.OPTION "FALSCH eingibt.

Hier noch der Vollständigkeit halber die Prüffunktion AUSSERHALB?

```
PR AUSSERHALB?  :XB  :YB
 WENN  :XB < :XBMIN  RUECKGABE  "WAHR
 WENN  :XB > :XBMAX  RUECKGABE  "WAHR
 WENN  :YB < :YBMIN  RUECKGABE  "WAHR
 WENN  :YB > :YBMAX  RUECKGABE  "WAHR
 RUECKGABE  "FALSCH
ENDE
```

Aufgabe 5.7: Schreibe eine vereinfachte Version der Prozedur ACHSEN mit den Befehlen AUFX und AUFY.

Nun zur Schaubild-Prozedur. Als Parameter soll sie den Namen der zu zeichnenden Funktion :FN, die Eckwerte :XMIN, :XMAX, :YMIN, :YMAX der Weltkoordinaten und die Schrittweite :H haben, mit der das Urbildintervall zu durchlaufen ist.

```
PR FUNKTIONS.SCHAUBILD :FN :XMIN :XMAX :YMIN :YMAX :H
 ACHSEN  :XMIN  :XMAX  :YMIN  :YMAX
 VOLLBILD  STIFTHOCH
 FUNKTIONS.SCHAUBILD.HP  :XMIN
 TEILBILD
ENDE

PR FUNKTIONS.SCHAUBILD.HP  :X
 WENN  :X > :XMAX  DANN RUECKKEHR
 LOKAL "XB
 SETZE "XB  TRANSX :X
 LOKAL "YB
 SETZE "YB  TRANSY  TUE WERT :FN
 WENN  AUSSERHALB? :XB :YB  DANN  STIFTHOCH ______
 ____  SONST  AUFXY :XB :YB  STIFTAB
 FUNKTIONS.SCHAUBILD.HP  ( :X + :H )
ENDE
```

Die Prozedur FUNKTIONS.SCHAUBILD dient im wesentlichen nur der "Ablaufsteuerung". Die eigentliche Arbeit wird von der Hilfsprozedur FUNKTIONS.SCHAUBILD.HP geleistet. Diese (endrekursive) Prozedur übernimmt die Variablenwerte :XMIN, :XMAX, :YMIN, :YMAX und :H von der aufrufenden Prozedur FUNKTIONS.SCHAUBILD. (Dasselbe gilt übrigens auch für die Transformationsfunktionen TRANSX und TRANSY). Mit den Befehlen STIFTAB (SA) und STIFTHOCH (SH) kann man festlegen, ob der Igel eine Spur hinter sich her zieht oder nicht. Nach dem Befehl STIFTHOCH hinterläßt der Igel keine Spur.

Um ganz deutlich zu machen, wie bestimmte Prozeduren aufzurufen sind, fügt man am besten eine kleine "Demo"-Prozedur hinzu; etwa so:

```
PR DEMO
 SETZE  "RAHMEN.OPTION  "WAHR
 LOKAL  "TESTFKT
 SETZE  "TESTFKT   [ SIN :X ]
 FUNKTIONS.SCHAUBILD  "TESTFKT (-20) 380 (-1.1) 1.1 0.1
ENDE
```

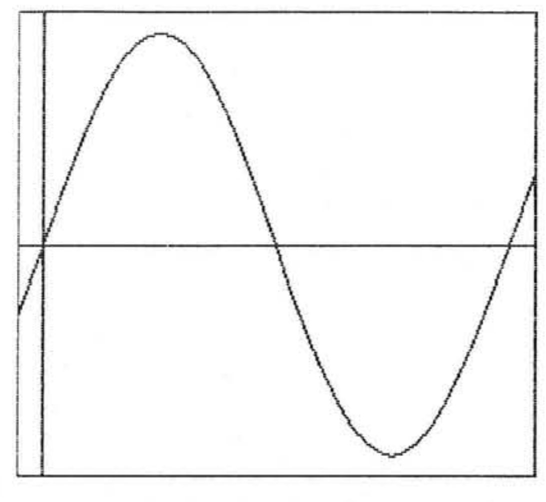


Bild 5.5: Die Sinusfunktion (im Gradmaß)

Aufgabe 5.8: Experimentiere weiter mit diesen Prozeduren. Setze die Bildschirmeckwerte so, daß du nacheinander vier kleine Funktionsschaubilder zeichnen kannst, die jeweils ein Viertel des Bildschirms ausfüllen; etwa so:

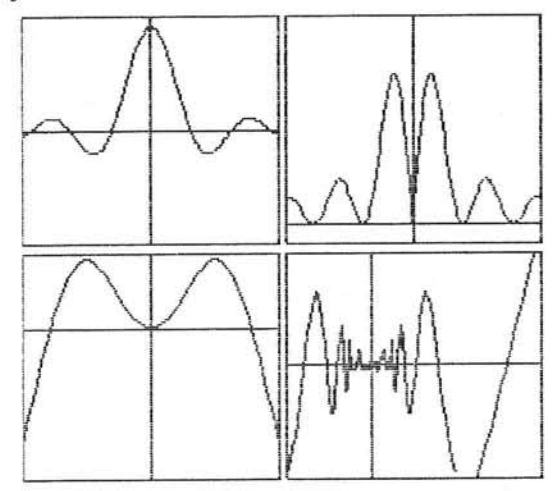


Bild 5.6: Mehrere Funktions-"Fenster"

Die bisher beschriebenen Prozeduren stellen nur ein Grundgerüst zum Zeichnen von Funktionsschaubildern dar. In den folgenden Aufgaben sind einige Anregungen zusammengestellt, wie du diese Prozeduren ausbauen und erweitern kannst.

Aufgabe 5.9: Erweitere die Prozedur FUNKTIONS.SCHAUBILD so, daß die Funktion entweder über ihren Namen oder als Funktionsliste eingegeben werden kann (siehe die Prozedur WERTETAFEL in Abschnitt 5.1).

Aufgabe 5.10: Schreibe Prüffunktionen
IN.ORDNUNG.WELTECKWERTE? bzw.
IN.ORDNUNG.BILDSCHIRMECKWERTE?
mit denen Fehler bei der Eingabe der jeweiligen Eckwerte entdeckt werden können. Zum Beispiel muß stets :XMIN kleiner als :XMAX sein usw., damit kein Unsinn herauskommt. Baue diese Prüffunktionen in die obigen Arbeitsprozeduren ein.

Aufgabe 5.11: Baue eine Möglichkeit ein, die Achsen und das Funktionsschaubild in verschiedenen (frei wählbaren) Farben zu zeichnen.

Aufgabe 5.12: Baue eine Möglichkeit ein, mehrere Funktionsschaubilder in ein Koordinatensystem auf den Bildschirm zu zeichnen.

Wenn man bei der linearen Funktion $f(x) = m*x + b$ den Parameter m (bzw. b) systematisch verändert und jedesmal ein Schaubild der Funktion in dasselbe Koordinatensystem zeichnet, dann nennt man diese Serie von Schaubildern eine **Funktionsschar.** Auch für quadratische und andere Funktionen kann man auf entsprechende Weise Funktionsscharen erhalten.

Aufgabe 5.13: Schreibe Prozeduren
FUNKTIONSSCHAR.LINEAR
FUNKTIONSSCHAR.QUADRATISCH
FUNKTIONSSCHAR.SINUS
Plane sorgfältig, was für Eingabeparameter diese Prozeduren haben sollten. Schreibe weitere Prozeduren zum Zeichnen von Funktionsscharen.

Aufgabe 5.14: (Ein kleines Projekt)
Schreibe eine Prozedur zum Skalieren der Achsen (in sinnvollen Einheiten) und baue sie in die Prozedur ACHSEN ein.

Aufgabe 5.15: Stelle die folgenden Funktionen in einem Schaubild dar (x-Bereich etwa: -0.5 bis 7).

```
(a)   F :X   =  SIN :X * 180 / 3.14159
(b)   F1 :X  =  :X
(c)   F2 :X  =  :X - :X*:X*:X / (2*3)
(d)   F3 :X  =  :X - :X*:X*:X / (2*3) +
                   :X*:X*:X*:X*:X / (2*3*4*5)
(e)   F4 :X  =  :X - :X*;X*:X / (2*3) +
                   :X*:X*:X*:X*:X / (2*3*4*5) -
                   :X*:X*:X*:X*:X*:X*:X / (2*3*4*5*6*7)
(f)   F5 :X  =  :X - :X*:X*:X / (2*3) +
         :X*:X*:X*:X*:X / (2*3*4*5) -
         :X*:X*:X*:X*:X*:X*:X / (2*3*4*5*6*7) +
         :X*:X*:X*:X*:X*:X*:X*:X*:X / (2*3*4*5*6*7*8*9)
```

Wie geht es weiter mit F6, F7, ... ? Teste deine Vermutung durch das Zeichnen weiterer Schaubilder. Erweitere den Bereich auf der x-Achse gegebenenfalls nach rechts.

Automatisches Nachladen von Prozeduren: Diese Stelle ist gut geeignet, um ein weiteres Beispiel für die Flexibilität von Logo anzusprechen. Ähnliche Dinge sind in kaum einer anderen Programmiersprache (außer Lisp) möglich.

Die Prozeduren TRANSX, TRANSY, MAXIMALER.RAHMEN!, RAHMEN.DIALOG, RAHMEN, ACHSEN und AUSSERHALB? stellen (eventuell noch ergänzt durch weitere Dienstprozeduren wie zum Beispiel SKALIERUNG) eine in sich abgeschlossene Gruppe von Prozeduren dar, die von der Prozedur FUNKTIONS.SCHAUBILD aus benutzt wird. Es ist aber denkbar, daß die Prozedurgruppe um die Prozedur ACHSEN auch noch von anderen Prozeduren aus benutzt wird; so zum Beispiel von der Prozedur HISTOGRAMM, die wir später noch schreiben wollen. Es ist also sinnvoll, die Prozeduren TRANSX, ..., ACHSEN und AUSSERHALB? unabhängig von der Prozedur FUNKTIONS.SCHAUBILD abzuspeichern. Bezogen auf das, was wir bisher in diesem Abschnitt geschrieben haben, können wir dies entweder von Hand oder mit der folgenden Prozedur tun:

```
PR ACHSENUMGEBUNG.ABSPEICHERN
 ZEIGE TITEL      ; NUR DES UEBERBLICKS WEGEN
 BEWAHRE  "HILFSDATEI
 VERGISS  FUNKTIONS.SCHAUBILD
 VERGISS  FUNKTIONS.SCHAUBILD.HP
 VERGISS  DEMO
 ; EVENTUELL WEITERE PROZEDUREN LOESCHEN, DIE
 ; NICHT ZUR ACHSEN-UMGEBUNG GEHOEREN
 ZEIGE TITEL      ; NUR DES UEBERBLICKS WEGEN
 BEWAHRE  "ACHSENDATEI
 ADE
 LADE  "HILFSDATEI
 VERGISS  TRANSX
 VERGISS  TRANSY
 VERGISS  MAXIMALER.RAHMEN!
 VERGISS  RAHMEN.DIALOG
 VERGISS  RAHMEN
 VERGISS  ACHSEN
 VERGISS  AUSSERHALB?
 ZEIGE TITEL       ; NUR DES UEBERBLICKS WEGEN
 ; EVENTUELL WEITERE PROZEDUREN LOESCHEN, DIE
 ; ZUR ACHSEN-UMGEBUNG GEHOEREN
 BEWAHRE  "FS.DATEI      ; FS: FUNKTIONS.SCHAUBILD
 ; EVENTUELL NOCH:  VERGISSDATEI "HILFSDATEI
ENDE
```

Der Logo-Grundbefehl VERGISS XYZ bewirkt, daß die Prozedur XYZ aus dem Arbeitsspeicher gelöscht wird. (Der Prozedurname XYZ darf ausnahmsweise in diesem Zusammenhang nicht gequotet werden). Der Strichpunkt (Semikolon) dient wieder zur Kennzeichnung von Kommentaren.

Jetzt sind der Achsen-Teil und der Funktionsschaubild-Teil unserer Prozeduren sauber auf der Diskette getrennt. Wenn wir nach dem Laden von Logo Funktionsschaubilder zeichnen wollen, müssen wir die beiden Dateien FS.DATEI und ACHSENDATEI laden.

Abschließend soll hier eine Möglichkeit gezeigt werden, wie beim Lauf der Prozedur FUNKTIONS.SCHAUBILD automatisch festgestellt werden kann, ob die Prozedur ACHSEN im Arbeitsspeicher vorhanden ist oder nicht und wie die ACHSENDATEI gegebenenfalls automatisch nachgeladen werden kann. Dabei wird auf fortgeschrittene Elemente der Listenprogrammierung zurückgegriffen, deren Entwicklung den Rahmen dieses Buches überschreiten würde. Dieser Themenkreis ist ausführlich in dem Buch *"Programmieren lernen mit Logo"*, Hanser Verlag, (besonders Kapitel 8) beschrieben. Man kann die folgende Prozedur aber dennoch einfach abtippen und benutzen.

Der Aufruf PROZEDURNAME? :PN gibt WAHR oder FALSCH zurück, je nachdem ob eine Prozedur unter dem Namen :PN im Arbeitsspeicher vorhanden ist oder nicht.

```
PR PROZEDURNAME?  :PN
 RUECKGABE  ALLE? (LISTE? PL :PN)  (NICHT? LEER? PL :PN)
ENDE
```

```
PROZEDURNAME? "FUNKTIONS.SCHAUBILD
ERGEBNIS: WAHR
```

PROZEDURNAME? "XAVER
ERGEBNIS: FALSCH

PROZEDURNAME? "VORWAERTS
ERGEBNIS: FALSCH

PROZEDURNAME? "ACHSEN
ERGEBNIS: WAHR

Wenn wir jetzt folgendes als erste logische Zeile in die Prozedur FUNKTIONS.SCHAUBILD einfügen, dann wird die ACHSENDATEI im Bedarfsfall nachgeladen:

```
WENN NICHT? PROZEDURNAME? "ACHSEN DANN LADE "ACHSENDATEI
```

5.3 Polynome und das Horner-Schema

Ein *Polynom* ist eine Schreibfigur der Art:

$$a_n * x^n + a_{n-1} * x^{n-1} + ... + a_2 * x^2 + a_1 * x + a_0 \qquad (*)$$

Dabei sei n eine natürliche Zahl; a_0 , a_1, ... , a_n seien beliebige rationale (oder reelle) Zahlen. Man nennt sie die *Koeffizienten* des Polynoms. Ist a_n von Null verschieden, so hat das obige Polynom den *Grad* n. Als Kurzschreibweise für Polynome werden auch die Darstellungen p(x), q(x), usw. verwendet, wenn es auf die Koeffizienten im einzelnen nicht ankommt oder wenn sie vom Zusammenhang her klar sind.

Beispiele:

$$p(x) = 1.5 * x^3 - 12 * x^2 - 4.5 * x + 47 \quad \text{(Grad 3)}$$

$$q(x) = 3.5 * x^2 - 10 * x + 17 \quad \text{(Grad 2)}$$

$$r(x) = 0.9 * x + 2 \quad \text{(Grad 1)}$$

Wenn wir uns vorstellen, daß x eine Variable ist, für die (reelle) Zahlen eingesetzt werden können, dann läßt sich jedes Polynom als eine Funktion der Art

$$x \longrightarrow p(x)$$

deuten. Man sagt dann, das Polynom werde an der Stelle x *ausgewertet*. Wir wollen uns nun der Auswertung von Polynomen zuwenden. Bei der Auswertung von Polynomen der obigen Art kommt es nur auf die Koeffizienten an; dagegen ist es im Grunde unwichtig, ob wir x oder eine andere Variable benutzen. Wir wollen jedes Polynom deshalb kurz durch die Liste seiner Koeffizienten darstellen. In den obigen Beispielen etwa:

```
p : [ 1.5   -12   -4.5   47 ]
q : [ 3.5   -10    17 ]
r : [ 0.9    2 ]
```

Beachte: Die Additionszeichen werden nicht geschrieben. An Stelle einer Subtraktion schreiben wir eine Addition mit "umgekehrten" Vorzeichen entsprechend der Regel:

$$a - b = a + (-b)$$

Aufgabe 5.16: Schreibe eine Funktion
POLYNOMAUSWERTUNG :KOEFFIZIENTENLISTE :X
mit der du das durch die Koeffizientenliste gegebene Polynom an der Stelle :X auswerten kannst. Verwende dabei die obige Darstellungsform (*).

Das Horner-Schema

Wenn man Polynome etwas anders schreibt, lassen sie sich sehr viel besser auswerten. Dies soll am Beispiel des Polynoms

$$p(x) = a * x^3 + b * x^2 + c * x + d \qquad (D1)$$

gezeigt werden. Wir können dieses Polynom auch folgendermaßen schreiben:

$$p(x) = ((a * x + b) * x + c) * x + d \qquad (D2)$$

Aufgabe 5.17: Rechne nach, daß die rechten Seiten von (D1) und (D2) gleich sind.

Wir wollen einmal den Aufwand bei diesen beiden Berechnungsarten vergleichen.

Darstellung	Anzahl der Additionen	Anzahl der Multiplikationen
D1	3	6
D2	3	3

Aufgabe 5.18: (a) Begründe, warum der Vergleich bei allgemeinen Polynomen der Form (*) zu der folgenden Tabelle führt (n = Grad des Polynoms).

Darstellung	Anzahl der Additionen	Anzahl der Multiplikationen
D1	n	1 + 2 + 3 + ... + n
D2	n	n

(b) Überprüfe die folgende Gleichung anhand von Zahlenbeispielen für verschiedene Werte von n:
$1 + 2 + 3 + ... + n = n * (n+1) / 2$
(c) Für Liebhaber: Zeige, daß diese Gleichung allgemein gilt.
(d) Stelle die Funktionen
$f(n) = n$ und
$g(n) = n * (n+1) / 2$
im Schaubild dar und vergleiche ihr Wachstum (besonders für große Werte von n).

Die Auswertung des Polynoms nach der Darstellung (D2) wird als das **Horner-Schema** bezeichnet. Sie ist also mit erheblich weniger Aufwand verbunden als die Auswertung von (D1). Außerdem läßt sie sich sehr leicht beschreiben. Die Auswertungsfunktion soll HORNER heißen und als Eingabeparameter die Koeffizien-

tenliste :KL und die Auswertungsstelle :X besitzen. Die Darstellung (D2) läßt sich folgendermaßen in Logo umsetzen:

```
PR HORNER  :KL  :X
 WENN  LEER? :KL  DANN RUECKGABE 0
 RUECKGABE  ( LZ :KL )  +  :X * HORNER  ( OL :KL )  :X
ENDE
```

Beispiele:

```
HORNER [ 1 2 3 ] 5
ERGEBNIS: 38

HORNER [ 0.1  -6.07  109.95  4800  16266 ] 0.672
ERGEBNIS: 19539.43
```

Beim nächsten Problem muß gerundet werden. Da das **Runden** aber von eigenständigem Interesse ist, soll es hier zunächst unabhängig davon behandelt werden. Die folgende Funktion rundet einen eingegebenen Betrag :X auf den nächsten durch 30 ohne Rest teilbaren vollen Betrag ab, wenn :X nicht bereits durch 30 ohne Rest teilbar ist. Die dabei verwendete Grundfunktion INT schneidet einfach die Stellen hinter dem Komma ab:

```
INT 3.14
ERGEBNIS: 3

INT 7.0
ERGEBNIS: 7
```

```
PR ABRUNDEN.AUF.VIELFACHES.VON.30  :X
 RUECKGABE  30 * INT ( :X / 30 )
ENDE
```

```
ABRUNDEN.AUF.VIELFACHES.VON.30 42395.68
ERGEBNIS: 42390

ABRUNDEN.AUF.VIELFACHES.VON.30 25140
ERGEBNIS: 25140
```

Aufgabe 5.19: (a) Prüfe die letzten beiden Aufrufe von Hand nach.
(b) Schreibe eine Funktion ABRUNDEN.AUF.VIELFACHES.VON.60 :X
(c) Schreibe eine Funktion ABRUNDEN.AUF.VIELFACHES.VON :A :X

Aufgabe 5.20: In Paragraph 32a des Einkommensteuergesetzes 1983 ist festgelegt, wie die Einkommensteuer für das jeweilige zu versteuernde Einkommen zu berechnen ist (siehe unten). Schreibe eine Funktion Programm EKST :E, die für jedes Einkommen :E die entsprechende Einkommensteuer ausgibt.

§ 32 a

Einkommensteuertarif

(1) Die tarifliche Einkommensteuer bemißt sich nach dem zu versteuernden Einkommen. Sie beträgt vorbehaltlich der §§ 32b, 34 und 34b jeweils in Deutsche Mark

1. für zu versteuernde Einkommen bis 4 212 Deutsche Mark (Grundfreibetrag): 0;
2. für zu versteuernde Einkommen von 4 213 Deutsche Mark bis 18 000 Deutsche Mark:
   0,22 x – 926;
3. für zu versteuernde Einkommen von 18 001 Deutsche Mark bis 59 999 Deutsche Mark:
   ([(3,05 y – 73,76) y + 695] y + 2 200) y + 3 034;
4. für zu versteuernde Einkommen von 60 000 Deutsche Mark bis 129 999 Deutsche Mark:
   ([(0,09 z – 5,45) z + 88,13] z + 5 040) z + 20 018;
5. für zu versteuernde Einkommen von 130 000 Deutsche Mark an:
   0,56 x – 14 837.

„x" ist das abgerundete zu versteuernde Einkommen. „y" ist ein Zehntausendstel des 18 000 Deutsche Mark übersteigenden Teils des abgerundeten zu versteuernden Einkommens. „z" ist ein Zehntausendstel des 60 000 Deutsche Mark übersteigenden Teils des abgerundeten zu versteuernden Einkommens.

(2) Das zu versteuernde Einkommen ist auf den nächsten durch 54 ohne Rest teilbaren vollen Deutsche-Mark-Betrag abzurunden, wenn es nicht bereits durch 54 ohne Rest teilbar ist.

(3) Die zur Berechnung der tariflichen Einkommensteuer erforderlichen Rechenschritte sind in der Reihenfolge auszuführen, die sich nach dem Horner-Schema ergibt. Dabei sind die sich aus den Multiplikationen ergebenden Zwischenergebnisse für jeden weiteren Rechenschritt mit drei Dezimalstellen anzusetzen; die nachfolgenden Dezimalstellen sind fortzulassen. Der sich ergebende Steuerbetrag ist auf den nächsten vollen Deutsche-Mark-Betrag abzurunden.

5.4 Stellenwertsysteme

Wenn wir von der Zahl 6174 sprechen und nichts weiter dazu sagen, dann meinen wir die Darstellung im Zehnersystem, die folgendermaßen ausführlicher geschrieben werden kann:

$$6174 = 6 * 1000 + 1 * 100 + 7 * 10 + 4 * 1$$

oder

$$6174 = 6 * 10^3 + 1 * 10^2 + 7 * 10^1 + 4 * 10^0$$

Manchmal möchte man auch Zahlendarstellungen in anderen Zahlensystemen verwenden. Wer viel mit Computern arbeitet, kennt sicher die Darstellungen im **Zweiersystem** (auch *Binär-* oder *Dualsystem* genannt), im **Achtersystem** (*Oktalsystem*) oder im **Sechzehnersystem** (*Hexadezimalsystem*).

Nehmen wir einmal an, die Zahl 6174 sei eine Zahl im Achtersystem. Dann erhalten wir ihren Wert im Zehnersystem, indem wir den Ausdruck

$$6 * 8^3 + 1 * 8^2 + 7 * 8 + 4$$

berechnen. Es liegt nahe, diesen Ausdruck als Wert des Polynoms

$$6 * x^3 + 1 * x^2 + 7 * x + 4$$

an der Stelle $x = 8$ anzusehen und zur Auswertung die HORNER-Funktion zu verwenden.

```
HORNER [ 6 1 7 4 ] 8
ERGEBNIS: 3196   (dezimal)
```

Aufgabe 5.21: Stelle dir vor, die Zahl 6174 sei im Sechzehnersystem aufgeschrieben. Ermittle ihren Wert im Zehnersystem.

Wenn wir, etwas allgemeiner, sagen: "Die Zahl rsuvw ist im System mit der **Basis** b (kurz: b-System) geschrieben", dann bedeutet das in ausführlicherer Schreibweise:

$$\text{rsuvw} = r * b^4 + s * b^3 + u * b^2 + v * b + w \quad (\#)$$

Aufgabe 5.22: (a) Begründe, warum man im Zehnersystem nur die Ziffern 0, 1, 2, ..., 9 benötigt.
(b) Welche Ziffern benötigt man im Sechsersystem?
(c) Wie viele Ziffern benötigt man im b-System?

Mit dem Horner-Schema können wir Zahlen, die im b-System dargestellt sind, ins Zehnersystem übertragen. Wir wollen uns nun der umgekehrten Aufgabe zuwenden. Die Umwandlungsfunktion, mit der wir *vom Zehnersystem ins b-System* gelangen, soll BASIS heißen und als Eingabeparameter :B (für die neue Basis) und :N (für die umzuwandelnde Zahl) besitzen. Die Darstellung im b-System soll in Listenform erfolgen, also etwa so:

13 (dezimal) = [1 1 0 1] (binär)

Das Verfahren soll am Beispiel der Zahl n erläutert werden, die im b-System die Darstellung rsuvw (siehe (#)) haben möge. Wir stehen also vor der Aufgabe, die "Ziffern" r, s, u, v und w zu bestimmen. Wir schreiben die Zahldarstellung zunächst ein klein wenig um:

$$n = \text{rsuvw} = (r * b^3 + s * b^2 + u * b + v) * b + w \quad (\#\#)$$

Aufgabe 5.23: Begründe: w ist der Divisionsrest bei der Ganzzahldivision von n durch b; in "Logo": w = REST n b = REST rsuvw b

Die letzte Ziffer (hier w) läßt sich also leicht bestimmen. Wie geht es weiter? Die vorletzte Ziffer von rsuvw (hier v) ist die letzte Ziffer der Zahl rsuv. Für rsuv gilt:

$$n = \text{rsuv} * b + w$$

Aufgabe 5.24: Begründe: rsuv = DIV n b

Um die Darstellung der Zahl n im b-System zu bekommen, müssen wir also einfach an die entsprechende Darstellung der Zahl

DIV n b

die durch

REST n b

gegebene "Ziffer" anhängen.

```
PR BASIS  :B  :N
 WENN  :N = 0   DANN  RUECKGABE  [ ]
 RUECKGABE MITLETZTEM (REST :N :B) (BASIS :B (DIV :N :B))
ENDE
```

```
BASIS 8 6174
ERGEBNIS: [ 1 4 0 3 6 ]
```

BASIS 16 6174
ERGEBNIS: [1 8 1 14]

Beachte: Im letzten Beispiel ist das Symbol 14 als *eine* Ziffer des Sechzehner-Systems anzusehen. Im Sechzehner-System benötigen wir 16 Ziffern. So, wie unsere BASIS-Funktion geschrieben ist, werden als Ziffern die folgenden durch Leerzeichen getrennten 16 Symbole verwendet:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Da wir unser Ergebnis als Liste geschrieben haben, sind die einzelnen Ziffern automatisch sauber getrennt. Bei der sonst eher üblichen Schreibweise von Zahlen in Form von Zahl-"Wörtern" kann es aber Probleme geben. Denn bei der Darstellung im Sechzehnersystem könnte die Zahl 111 zum Beispiel:

```
1 * 256  +  1 * 16  +   1     ( = 273 dezimal)  oder
           11 * 16  +   1     ( = 177 dezimal)  oder
            1 * 16  +  11     ( =  27 dezimal)
```

bedeuten, je nachdem, wo die Trennung der Ziffern anzusetzen ist. Man schreibt jede Ziffer deshalb gern als einzelnes Zeichen. Eine Standardmethode besteht darin, daß man nach der Ziffer 9 einfach die Buchstaben A, B, C, usw. in alphabetischer Reihenfolge benutzt. Im Sechzehnersystem hat man dann die "Ziffern":

0 1 2 3 4 5 6 7 8 9 A B C D E F

Die obigen drei Möglichkeiten zur Aufteilung der Zahl 111 würden jetzt 111 bzw. B1 bzw. 1B lauten. Die Darstellung wäre somit eindeutig.

Aufgabe 5.25: Der Aufruf BASIS :B :N liefert die leere Liste, falls :N = 0 ist. Es wäre besser, wenn dieser Aufruf zum Ergebnis [0] führen würde. Ändere die BASIS-Funktion entsprechend ab. (Ansonsten soll aber nichts verändert werden).

Aufgabe 5.26: Bis zu welcher maximalen Basis kommt man mit der "Buchstaben-als-Ziffern"-Methode?

Aufgabe 5.27: (a) Ermittle die Darstellung der Zahl 6174 (dezimal) im Zweier-, Dreier-, ... , Sechzigersystem.
(b) Stelle eine Tabelle der folgenden Art auf:

```
  Basis    !  Anzahl der "Ziffern"
---------------------------------
    2      !    13
    3      !     8
   ...     !    ...
   10      !     4
   ...     !    ...
   60      !     3
```

(c) Stelle die Tabelle im (x/y)-Schaubild dar.

Die Funktionen HORNER und BASIS sind genau die Umkehrungen voneinander; was die eine tut, macht die andere rückgängig. Wir können dies leicht im Dialogbetrieb überprüfen:

HORNER (BASIS 3 975) 3
ERGEBNIS: 975

BASIS 6 (HORNER [5 1 4 2 3] 6)
ERGEBNIS: [5 1 4 2 3]

Die runden Klammern dienen nur der optischen Gliederung.

Aufgabe 5.28: Verfolge die letzten beiden Aufrufe im Protokoll-Modus.

Die Hexadezimaldarstellung

Die Darstellung im Sechzehnersystem mit der "Buchstabenmethode" wird meist als die *Hexadezimaldarstellung* bezeichnet. Wir wollen zum Schluß dieses Abschnitts noch eine Prozedur zur Anpassung unserer Ergebnisse an die Buchstabenmethode behandeln. Das Prinzip ist einfach: wir müssen jeweils nur 10 gegen A, 11 gegen B,...,15 gegen F austauschen. Dazu legen wir eine Tabelle als Paarliste an:

```
PR HEXTABELLE
 RUECKGABE  [ [ 10 A ]  [ 11 B ]  [ 12 C ]  ________
 _________    [ 13 D ]  [ 14 E ]  [ 15 F ]  ]
ENDE
```

Die folgende Funktion TAUSCH tauscht die "Ziffern" 10, 11, 12, 13, 14 und 15 gegen die "Ziffern" A, B, C, D, E und F aus. Alle anderen Eingaben werden unverändert zurückgegeben.

```
PR TAUSCH  :X  :PAARLISTE
 WENN  LEER?  :PAARLISTE  RUECKGABE :X
 WENN  :X = ERSTES ERSTES :PAARLISTE  DANN ____
 _________  RUECKGABE  LETZTES ERSTES :PAARLISTE
 RUECKGABE  TAUSCH  :X  OHNEERSTES :PAARLISTE
ENDE
```

Beispiele:

TAUSCH 14 HEXTABELLE
ERGEBNIS: E

TAUSCH 3 HEXTABELLE
ERGEBNIS: 3

Da das Ergebnis der Funktion BASIS als Liste ausgegeben wird, benötigen wir noch eine Funktion, die alle Elemente der Liste tauscht. Wir nennen sie LISTENTAUSCH.

```
PR LISTENTAUSCH  :L  :PAARLISTE
 WENN LEER? :L  RUECKGABE  :L
 RUECKGABE  MITERSTEM  TAUSCH  ERSTES :L :PAARLISTE ____
 ______________  LISTENTAUSCH OHNEERSTES :L :PAARLISTE
ENDE
```

Beispiel:

LISTENTAUSCH [4 10 12 3 15] HEXTABELLE
ERGEBNIS: [4 A C 3 F]

Um Zahlen ins Sechzehnersystem mit Buchstabendarstellung umzuwandeln, müssen wir also zum Beispiel schreiben:

LISTENTAUSCH BASIS 16 6174 HEXTABELLE
ERGEBNIS: [1 8 1 E]

Wenn man viel im Sechzehnersystem arbeitet, ist dieser lange Aufruf lästig. Wir schreiben deshalb einen Kurzaufruf.

```
PR HEX  :N
 RUECKGABE  LISTENTAUSCH  BASIS  16  :N  HEXTABELLE
ENDE
```

HEX 509
ERGEBNIS: [1 F D]

Schließlich wollen wir von der Listendarstellung wieder zur gewöhnlichen Zahlwortdarstellung zurückkehren. Wir benötigen dazu eine Funktion, welche die einzelnen Listenelemente einfach aneinanderhängt und so das Zahlwort bildet. Wir wollen diese Funktion mit dem Namen MACHE.WORT versehen.

```
PR MACHE.WORT :L
 WENN  LEER? :L  DANN RUECKGABE  "
 RUECKGABE  ( WORT ERSTES :L  MACHE.WORT OHNEERSTES :L )
ENDE
```

MACHE.WORT [A 1 B 2 C 3 X 107 Y]
ERGEBNIS: A1B2C3X107Y

Auch dies verpacken wir noch in einer Funktion:

```
PR HEXWORT :N
 RUECKGABE  MACHE.WORT  HEX :N
ENDE
```

Beispiele:

HEXWORT 59
ERGEBNIS: 3B

HEXWORT 255
ERGEBNIS: FF

Um anzudeuten, daß eine Zahl im Hexadezimalsystem gegeben ist, verwendet man gelegentlich auch das $ - Zeichen. So ist zum Beispiel $ FF dieselbe Zahl wie 255 (im Zehnersystem). Man kann das $-Zeichen auch als Funktionsnamen deuten. Die zugehörige Funktion macht aus dem Zahlwort in der Hexadezimaldarstellung ein Zahlwort in der Dezimaldarstellung.

Aufgabe 5.29: Schreibe eine Funktion $, mit der du Hexadezimalzahlen (Buchstabenform) in Dezimalzahlen umwandeln kannst. Zum Beispiel:
$ "FF
ERGEBNIS: 255
Schreibe dazu Funktionen, mit denen du den Entwicklungsprozess, der zur Funktion HEXWORT geführt hat, rückwärts durchlaufen kannst.

Aufgabe 5.30: Die Funktion MACHE.WORT ist noch nicht gegen falsche Eingaben abgesichert. Beim Aufruf MACHE.WORT [A B C [D E] F] bringt Logo eine Fehlermeldung. Dies liegt daran, daß die Eingabeliste zu MACHE.WORT nicht nur Wörter, sondern ihrerseits wieder Listen enthält.
(a) Überprüfe derartige Fälle und fange sie mit einer Fehlermeldung ab.
(b) Schreibe eine Funktion MACHE.WORT.NEU, die auch noch Teillisten der Ausgangsliste in das zu bildende Wort einbettet; etwa so:

```
MACHE.WORT.NEU [ A B C [ D E ] F ]
ERGEBNIS: ABCDEF
```

Aufgabe 5.31: (Ein kleines Projekt für Liebhaber)
(a) Schreibe eine Prozedur SYSTEMBRUCH :B :DEC, mit der du den Bruch :DEC (dargestellt im Zehnersystem) als "System"-Bruch im System zur Basis :B darstellen kannst.
(b) Erkunde, wie es sich mit den Perioden in anderen Stellenwertsystemen verhält. Zum Beispiel ist der Bruch 1 / 3 im Zehnersystem periodisch, im Dreiersystem aber nicht, denn dort schreibt sich dieser Bruch als 0,1.

Die Funktionen dieses Abschnitts waren meist voll rekursiv. In diesem Fall ist das weniger schlimm, da in der Regel nur wenige Selbstaufrufe vorkommen. Der Speicher läuft also bei diesen Funktionen praktisch nicht voll.

Aufgabe 5.32: Schreibe (zur Übung oder auch "für alle Fälle") endrekursive Versionen von allen Funktionen dieses Abschnitts.

Kapitel 6: Simulation zufälliger Ereignisse

Wie wir schon in Kapitel 3, Abschnitt 3.4 über den größten gemeinsamen Teiler gesehen haben, gibt Logo beim Aufruf ZUFALLSZAHL :N (kurz ZZ :N) eine zufällig ausgewählte ganze Zahl zwischen 0 und :N-1 (jeweils einschließlich) zurück. Ein Beispiel:

```
WH 5 [ DZ ZZ 100 ]
57
12
3
78
89
```

Es ist durchaus eine sehr schwierig zu beantwortende Frage, warum ein Computer, bei dem doch alles vollständig exakt abläuft, Zufälliges produzieren kann, wie er das tut und ob man sich darauf verlassen kann, daß es sich wirklich um Zufälliges handelt. Die Untersuchung dieser Frage würde hier jedoch viel zu weit führen. Im folgenden stellen wir uns auf den Standpunkt, daß der Computer tatsächlich "echte" Zufallszahlen produziert.

6.1 Würfel

Mit dem ZUFALLSZAHL-Grundbefehl können wir schon einfache Zufallsgeräte **simulieren** ("simulieren" heißt etwa soviel wie "nachspielen").

```
PR WUERFELZAHL
 RUECKGABE  1 + ZUFALLSZAHL 6
ENDE
```

Der Aufruf ZUFALLSZAHL 6 produziert eine ganze Zahl zwischen 0 und 5; durch Addition von 1 erhalten wir gerade eine "Würfelzahl". Ein Beispiel im Dialogbetrieb:

```
WH 50 [ DR WUERFELZAHL ]
51435432345654124623614564234552341123623442426643
```

Aufgabe 6.1: Schreibe je eine Funktion TETRAEDERZAHL (Viererwürfel), OKTAEDERZAHL (Achterwürfel), DODEKAEDERZAHL (Zwölferwürfel) und IKOSAEDERZAHL (Zwanzigerwürfel).

Aufgabe 6.2: Schreibe eine Funktion ROULETTE.

In der folgenden Prozedur wird fortlaufend gewürfelt, und dabei wird die Anzahl der Einsen, Zweien, ... , Sechsen gezählt.

```
 1:  PR WUERFEL.STATISTIK
 2:   LOKAL "A1    SETZE "A1  0
 3:   LOKAL "A2    SETZE "A2  0
 4:   LOKAL "A3    SETZE "A3  0
 5:   LOKAL "A4    SETZE "A4  0
 6:   LOKAL "A5    SETZE "A5  0
 7:   LOKAL "A6    SETZE "A6  0
 8:   LOKAL "W
 9:   SCHLEIFENBEGINN:
10:   SETZE "W  WUERFELZAHL
11:   SETZE ( WORT "A :W )    ( WERT ( WORT "A :W ) ) + 1
12:   ( DZ  :A1  :A2  :A3  :A4  :A5  :A6 )
13:   GEHE  "SCHLEIFENBEGINN
14:  ENDE
```

Diese Prozedur zeigt, wie flexibel das Variablenkonzept von Logo ist. Das Entscheidende geschieht dabei in Zeile 11. Nehmen wir einmal an, eine 5 wurde gewürfelt. Der Wert von W (:W) ist also 5. Dann ist (WORT "A :W) dasselbe wie A5. In Zeile 11 wird also zunächst der Wert von A5 ermittelt, um eins erhöht und wieder der Variablen A5 zugewiesen. Bei jeder anderen Würfelzahl geht es entsprechend. Zeile 12 ist wieder ein Beispiel dafür, wie man mehrere Dinge mit einem Druckbefehl ausdrucken kann.

Ein Beispiel:

```
WUERFEL.STATISTIK
0    1    0    0    0    0
0    1    0    0    0    1
0    2    0    0    0    1
0    2    1    0    0    1
0    2    1    0    1    1
0    2    1    0    2    1
1    2    1    0    2    1
1    3    1    0    2    1
  ...
```

Aufgabe 6.3: Schreibe eine Prozedur ROULETTE.STATISTIK.

6.2 Glücksräder

Das folgende *Glücksrad* bringt Erfolg (1) mit der Wahrscheinlichkeit p und Mißerfolg (0) mit der Wahrscheinlichkeit q = 1 - p.

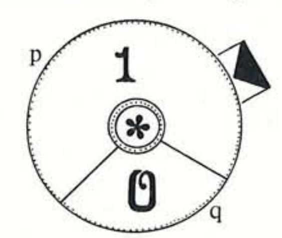


Bild 6.1: Glücksrad

Man kann das Glücksrad folgendermaßen in Logo simulieren:

```
PR GLUECKSRAD  :P
 WENN  ( ZZ 1000 ) / 1000  < :P  DANN RUECKGABE 1
 RUECKGABE 0
ENDE
```

WH 20 [DR GLUECKSRAD 0.3]
01001100001010100 10

Aufgabe 6.4: (a) Schreibe eine Prozedur GLUECKSRAD.STATISTIK :P, mit der du die produzierten Nullen und Einsen zählen kannst.
(b) Setze die Anzahl der Einsen ins Verhältnis zur Anzahl aller Versuche. Welcher Zahl müßte dieser Wert entsprechen?

6.3 Krumme Nägel

Hans Hammer will sich ein Bücherregal bauen. Dazu muß er einige Bretter zusammennageln. Seiner Planskizze entnimmt er, daß er an 100 Stellen nageln muß. Also braucht er 100 Nägel. Doch halt: von ähnlichen Bastelarbeiten her weiß er, daß er durchschnittlich 20 % der Nägel krumm schlägt und jeweils durch einen neuen Nagel ersetzen muß. Wie viele Nägel braucht er also wirklich? Da gerade seine Freunde Siegfried Schnellsprecher und Gustav Großsprecher bei Hans zu Besuch sind, bespricht er das Problem mit ihnen.

Aufgabe 6.5: Siegfried antwortet sofort: "Du brauchst 120 Nägel". Was wird Siegfried auf die Frage "Warum?" wohl antworten?

Aufgabe 6.6: Gustav hat eine andere Antwort: "Du brauchst 125 Nägel". Was wird wohl Gustavs Begründung sein?

Hans verläßt sich auf Gustavs Antwort. Er kauft 125 Nägel und fängt an, das Regal zu bauen. Doch er hat Pech. Als er alle Nägel verbraucht hat, fehlen ihm noch 4 Stück. Er zählt die krummgeschlagenen Nägel: 29 Stück. So ein Ärger! Wie viele Nägel hätte Hans eigentlich kaufen sollen?

Wir können versuchen, die gesuchte Zahl mit Hilfe einer Simulation zu schätzen. Die Prozedur NAEGEL :N :P.KRUMM hat als Eingabe die Anzahl :N der einzuschlagenden Nägel (im obigen Fall 100) und die Wahrscheinlichkeit :P.KRUMM, einen Nagel krummzuschagen. Für jeden richtig geschlagenen Nagel wird ein X, für jeden krummgeschlagenen ein "-" (also ein Strich) ausgedruckt. Zum Schluß wird die Anzahl der benötigten Nägel ausgegeben.

```
PR NAEGEL  :N  :P.KRUMM
 LOKAL  "GESAMT    SETZE  "GESAMT  0
 NAEGEL.HP  :N  :P.KRUMM   0
 RUECKGABE  :GESAMT
ENDE

PR NAEGEL.HP  :N  :P.KRUMM  :GUT
 WENN  :GUT = :N   DANN   DZ  "  RUECKKEHR
 PRUEFE GUTER.SCHLAG?  :P.KRUMM
 WENNWAHR  SETZE  "GUT  ( :GUT + 1 )  DR "X
 WENNFALSCH  DR "-
 SETZE  "GESAMT  ( :GESAMT + 1 )
 NAEGEL.HP  :N  :P.KRUMM  :GUT
ENDE

PR GUTER.SCHLAG?  :P.KRUMM
 RUECKGABE  NICHT? ( ( ZZ 1000 ) / 1000  <  :P.KRUMM )
ENDE
```

Ein Beispiel:

```
NAEGEL 100 20
XX-XXX-X--XXXXXXXX-XXX-XX-X-XXXXXX-XXXXX
XX-XXX--XX-XXX-XXXXX-XXXX-XXXXXXXX-XXX-X
XXXX-XX-X-XXXXXXX-XXX-XX-XXX-XXXXXX-XX-X
XX-X-XXXX
ERGEBNIS:  129
```

Mit einem einzigen Lauf dieser Prozedur kann Hans noch nicht viel anfangen. Er muß den Lauf mehrmals wiederholen, um eine ganze Reihe von Werten zu erhalten, aus denen er dann seine Schlüsse ziehen kann. Die folgende Funktion führt den Versuch mehrmals durch und speichert die Einzelergebnisse jeweils in einer Ergebnisliste ab, die zum Schluß als Funktionswert zurückgegeben wird.

```
PR NAEGEL.VERSUCHSREIHE  :N  :P.KRUMM  :LAENGE
 LOKAL  "ERGEBNISLISTE
 SETZE  "ERGEBNISLISTE  [ ]
 NAEGEL.VERSUCHSREIHE.HP  :N  :P.KRUMM  :LAENGE  1
 RUECKGABE  :ERGEBNISLISTE
ENDE

PR NAEGEL.VERSUCHSREIHE.HP  :N  :P.KRUMM  :LAENGE  :I
 WENN :I > :LAENGE  DANN RUECKKEHR
 ( DZ  "VERSUCH  "NR.  :I )
 SETZE  "ERGEBNISLISTE  MITLETZTEM
         ( NAEGEL  :N  :P.KRUMM )   :ERGEBNISLISTE
 NAEGEL.VERSUCHSREIHE.HP  :N  :P.KRUMM  :LAENGE  (:I+1)
ENDE
```

Es ist noch ganz sinnvoll, vor dem Lauf der Versuchsreihe alle Druckbefehle aus NAEGEL.HP herauszunehmen. (Wenn man sie später wieder braucht, dann kann man das auch dadurch erreichen, daß man vor die Druckbefehle ein Semikolon setzt. Der Rest der Zeile wird dann als Kommentar gewertet und nicht ausgeführt).

```
NAEGEL.VERSUCHSREIHE  100  0.2  20
ERGEBNIS:  [ 125 128 131 129 114 127 129 134 126 137
             127 127 132 120 123 125 128 124 132 129 ]
```

Aufgabe 6.7: Baue einige weitere "Zähler" in die Prozedur NAEGEL.VERSUCHSREIHE ein; zum Beispiel:
(a) Bei wie vielen Versuchen wird die Gesamtzahl 125 Überschritten?
(b) Wie oft wird die Gesamtzahl 130 (bzw. 140; 150) überschritten?
(c) Setze die absoluten Zahlen ins Verhältnis zur Anzahl der Versuche.

Aufgabe 6.8: Starte eine längere Versuchsreihe mit dem Ziel, eine möglichst kleine Nägelzahl :N (an Stelle von 125) herauszufinden, die höchstens in
(a) 5 %
(b) 1 %
aller Versuche überschritten wird.

6.4 Häufigkeitstabellen

Die von der Versuchsreihe produzierte **Urliste** ist ein erster Schritt zur Beantwortung der gestellten Frage. Die ermittelten Daten liegen aber in einer noch ziemlich ungeordneten Form vor. Dies stört besonders dann, wenn die Urliste umfangreicher ist als im obigen Fall. Die Darstellung der obigen Liste in der folgenden Form wäre übersichlicher:

```
114  1
120  1
123  1
124  1
125  2
126  1
127  3
128  2
129  3
131  1
132  2
134  1
137  1
```

Tabelle 6.1: Häufigkeitstabelle

Diese Darstellung bezeichnet man als **Häufigkeitstabelle.** Wir wollen jetzt eine Prozedur(-gruppe) schreiben, die aus einer ungeordneten (Ur-) Liste eine Häufigkeitstabelle erstellt. Die Tabelle wird als Liste von Paarlisten zurückgegeben. Diese Prozeduren stützen sich wieder ganz massiv auf das außerordentlich flexible Variablenkonzept von Logo.

```
 1: PR HAEUFIGKEITSTABELLE  :URLISTE
 2:  LOKAL  "HAEUFIGKEITSTABELLE
 3:  SETZE  "HAEUFIGKEITSTABELLE  [ ]
 4:  LOKAL  "MIN   SETZE  "MIN  ERSTES :URLISTE
 5:  LOKAL  "MAX   SETZE  "MAX  ERSTES :URLISTE
 6:  LOKAL  "N
 7:  LOKAL  "FELDNAME
 8:  HAEUFIGKEITSTABELLE.HP  :URLISTE
 9:  RUECKGABE  :HAEUFIGKEITSTABELLE
10: ENDE

11: PR HAEUFIGKEITSTABELLE.HP  :URLISTE
12:  WENN LEER?  :URLISTE  DANN ____________________________
13:  ___  HAEUFIGKEITSTABELLE.SAMMLUNG  :MIN  RUECKKEHR
14:  SETZE  "N  ERSTES :URLISTE
15:  WENN :N < :MIN  DANN  SETZE  "MIN  :N
16:  WENN :N > :MAX  DANN  SETZE  "MAX  :N
17:  SETZE  "FELDNAME  ( WORT "A  :N )
18:  PRUEFE  NAME?  :FELDNAME
19:  WENNFALSCH  LOKAL :FELDNAME  SETZE :FELDNAME  1
20:  WENNWAHR  SETZE :FELDNAME  ( WERT :FELDNAME ) + 1
21:  HAEUFIGKEITSTABELLE.HP  ( OHNEERSTES :URLISTE )
22: ENDE

23: PR HAEUFIGKEITSTABELLE.SAMMLUNG  :I
24:  WENN  :I > :MAX  DANN  RUECKKEHR
25:  SETZE  "FELDNAME  ( WORT "A  :I )
26:  PRUEFE  NAME?  :FELDNAME
27:  WENNWAHR SETZE "HAEUFIGKEITSTABELLE  MITLETZTEM ___
28:  __ (LISTE :I  WERT :FELDNAME)  :HAEUFIGKEITSTABELLE
39:  HAEUFIGKEITSTABELLE.SAMMLUNG  ( :I + 1 )
30: ENDE
```

Ein Beispiel:

```
HAEUFIGKEITSTABELLE  NAEGEL.VERSUCHSREIHE  100  0.2  50
ERGEBNIS:
__ [ [ 113 1 ] [ 116 2 ] [ 117 2 ] [ 118 2 ] [ 119 2 ] __
__   [ 120 6 ] [ 121 2 ] [ 122 5 ] [ 123 7 ] [ 124 4 ] __
__   [ 125 1 ] [ 126 3 ] [ 127 1 ] [ 128 2 ] [ 129 1 ] __
__   [ 130 2 ] [ 131 2 ] [ 133 1 ] [ 135 1 ] [ 136 1 ] __
__   [ 137 2 ]  ]
```

Aufgabe 6.9: Schreibe eine Prozedur zum Ausdruck von Paarlisten (insesondere Häufigkeitstabellen) in "kosmetischer" Darstellung.

Erläuterungen zur Prozedurgruppe HAEUFIGKEITSTABELLE: Die Zeilen 1, 2 und 3 zeigen, daß derselbe Name für Prozeduren und Daten (Variablen) verwendet werden kann, ohne daß Verwechslungen vorkommen. In den Variablen MIN und MAX wird jeweils das Minimum bzw. Maximum der Urlistenwerte gespeichert.

Die entscheidenden Zeilen 17 bis 20 sollen nun näher erläutert werden. Nehmen wir einmal an, der Wert von N (= ERSTES :URLISTE) ist 128. Dann wird in Zeile 17 der "Feldname" A128 zusammengebaut und der Variablen mit dem Namen FELDNAME zugeordnet. Der Wert von FELDNAME ist also A128. In Zeile 18 wird geprüft, ob es unter dem Namen A128 schon eine Variable gibt. Wenn nicht,

dann wird die Variable als lokale Variable erzeugt, und sie erhält den Wert 1 zugewiesen. (Zur Überprüfung, ob ein Variablenname vorhanden ist oder nicht, dient der Grundbefehl NAME?). Falls die Variable A128 jedoch schon (von einem früheren rekursiven Aufruf her) vorhanden war, wird ihr Wert in Zeile 20 um eins erhöht. Nach dem Ablauf der Hilfsprozedur HAEUFIGKEITSTABELLE.HP zeigt die lokale Variable A128 also an, wie oft die Zahl 128 in der Urliste vorgekommen ist. Entsprechendes gilt für alle anderen in der Urliste auftretenden Zahlen. Bezogen auf die in Tabelle 6.1 umgesetzte Versuchsreihe wurden durch den Lauf der Prozedur HAEUFIGKEITSTABELLE.HP die folgenden Variablen erzeugt:

```
A114  ( mit letztem Wert 1 )
A120  ( mit letztem Wert 1 )
A123  ( mit letztem Wert 1 )
A124  ( mit letztem Wert 1 )
A125  ( mit letztem Wert 2 )
A126  ( mit letztem Wert 1 )
A127  ( mit letztem Wert 3 )
A128  ( mit letztem Wert 2 )
A129  ( mit letztem Wert 3 )
A131  ( mit letztem Wert 1 )
A132  ( mit letztem Wert 2 )
A134  ( mit letztem Wert 1 )
A137  ( mit letztem Wert 1 )
```

Dies ist im wesentlichen schon eine mögliche Form der Häufigkeitstabelle. In der Prozedur HAEUFIGKEITSTABELLE.SAMMLUNG werden diese Variablen mit ihren Werten nur noch in die folgende Ergebnisliste umgesetzt:

```
  [ [ 114 1 ]  [ 120 1 ]  [ 123 1 ]  [ 124 1 ]  ______
___ [ 125 2 ]  [ 126 1 ]  [ 127 3 ]  [ 128 2 ]  ______
___ [ 129 3 ]  [ 131 1 ]  [ 132 2 ]  [ 134 1 ]  ______
___ [ 137 1 ] ]
```

Die Variable FELDNAME ist in diesen Prozeduren eine Variable, deren Wert seinerseits ein Variablenname (zum Beispiel A128) ist. Man kann sich das Verhältnis dieser Variablen folgendermaßen graphisch veranschaulichen:

FELDNAME ⟶ A128 ⟶ 2

Nach Ablauf der Prozedur HAEUFIGKEITSTABELLE sind alle zwischendurch erzeugten lokalen Variablen wieder verschwunden.

Aufgabe 6.10: Der Befehl ZEIGE NAMEN kann auch in Prozeduren verwendet werden. Bei seinem Aufruf listet Logo alle zum Zeitpunkt des Aufrufs bekannten Variablen mit ihren jeweiligen Werten auf. Baue den Befehl probehalber am Anfang in die Prozeduren HAEUFIGKEITSTABELLE.HP und HAEUFIGKEITSTABELLE.SAMMLUNG ein und beobachte, wie die Variablen erzeugt werden.

Die Art und Weise, wie das Problem der Häufigkeitstabelle hier gelöst wurde, hängt damit zusammen, daß Logo vorrangig den Datentyp der Liste unterstützt. Andere Programmiersprachen kennen meist keine Listen. Man würde ein derartiges Problem dort meist mit *Feldern* (englisch: *arrays*) lösen. Das Commodore Logo verfügt zur Zeit nicht über (eingebaute) Feld-Routinen. Man kann sie sich aber selbst schreiben (dies ist in dem Buch *"Programmieren lernen mit Logo"*, Hanser Verlag, beschrieben). Andere Logo-Versionen verfügen zum Teil auch über Felder.

Aufgabe 6.11: Falls du Erfahrungen mit Programmiersprachen hast, in der es Feld-Routinen gibt, dann schreibe eine Version der Prozedur HAEUFIGKEITSTABELLE für Felder.

Spätestens zu diesem Zeitpunkt drängt sich der Wunsch auf, die Tabelle 6.1 etwa folgendermaßen in ein Diagramm umzusetzen:

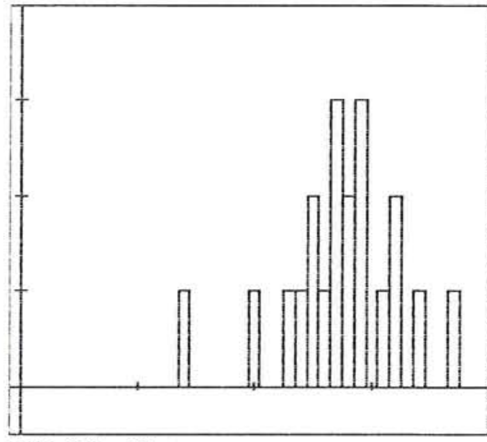


Bild 6.2: Histogramm

Derartige Schaubilder nennt man **Stabdiagramme** oder auch **Histogramme**. Wir werden in Abschnitt 6.8 Prozeduren zum Schreiben von Histogrammen kennenlernen.

6.5 Sortieren

Die zu der Urliste

```
    [ 125 128 131 129 114 127 129 134 126 137 ___
___   127 127 132 120 123 125 128 124 132 129 ]
```

gehörende Häufigkeitstabelle

```
    [ [ 114 1 ]  [ 120 1 ]  [ 123 1 ]  [ 124 1 ]  ______
___   [ 125 2 ]  [ 126 1 ]  [ 127 3 ]  [ 128 2 ]  ______
___   [ 129 3 ]  [ 131 1 ]  [ 132 2 ]  [ 134 1 ]  ______
___   [ 137 1 ] ]
```

kann auch dazu benutzt werden, um die Ausgangsliste zu sortieren. Die Hauptarbeit ist mit der Erstellung der Häufigkeitstabelle eigentlich schon erledigt. Wir brauchen die einzelnen Werte an Hand der Häufigkeitstabelle nur noch "auszuzählen" und hintereinander aufzuschreiben; etwa so:

```
1 mal die 114:    114
1 mal die 120:    120
1 mal die 123:    123
1 mal die 124:    124
2 mal die 125:    125 125
1 mal die 126:    126
3 mal die 127:    127 127 127
2 mal die 128:    128 128
3 mal die 129:    129 129 129
1 mal die 131:    131
2 mal die 132:    132 132
1 mal die 134:    134
1 mal die 137:    137
```

Die sortierte Liste ist also:

```
[ 114 120 123 124 125 125 126 127 127 127 ______
___ 128 128 129 129 129 131 132 132 134 137 ]
```

Die folgende Funktion führt den gerade beschriebenen Auszählvorgang durch:

```
PR AUSZAEHLEN  :PAARLISTE
 LOKAL  "ERGEBNIS
 SETZE  "ERGEBNIS  [ ]
 AUSZAEHLEN.HP  :PAARLISTE
 RUECKGABE  :ERGEBNIS
ENDE

PR AUSZAEHLEN.HP  :PAARLISTE
 WENN LEER? :PAARLISTE  DANN RUECKKEHR
 WIEDERHOLE (LZ ER :PAARLISTE)  [ SETZE "ERGEBNIS _____
 ______    MITLETZTEM ( ER ER  :PAARLISTE )  :ERGEBNIS ]
 AUSZAEHLEN.HP  OHNEERSTES  :PAARLISTE
ENDE
```

Nach diesen Vorarbeiten braucht die Funktion SORTIEREN jetzt nur noch die richtigen Hilfsfunktionen in der richtigen Reihenfolge aufzurufen.

```
PR SORTIEREN  :LISTE
 RUECKGABE  AUSZAEHLEN  HAEUFIGKEITSTABELLE  :LISTE
ENDE
```

Ein Beispiel:

```
SORTIEREN  [ 12 5 25 18 36 49 24 12 15  9 18 17 ]
ERGEBNIS:  [  5 9 12 12 15 17 18 18 24 25 36 49 ]
```

Dieser Sortieralgorithmus ist übrigens ziemlich schnell. Allerdings ist er in der vorliegenden Form zunächst nur auf Listen anwendbar, die aus ganzen Zahlen bestehen. Das Sortieren von Daten ist ein intensiv erforschtes Teilgebiet der Informatik, über das alleine man ein ganzes Buch schreiben könnte. Wir müssen uns hier mit diesem kleinen Beispiel begnügen.

6.6 Eine Roulette-Strategie

Gustav Gambler hat eine neue Strategie für das Roulette-Spiel entdeckt: Er setzt 1 DM auf ROT. Wenn ROT kommt, hat er 1 DM gewonnen. Wenn SCHWARZ kommt, setzt er 2 DM auf ROT. Wenn jetzt ROT kommt, hat er insgesamt 3 DM gesetzt und 4 DM ausgezahlt bekommen, also 1 DM gewonnen. Wenn nun beim zweiten Mal auch SCHWARZ erscheint, setzt er 4 DM auf ROT. Sein Gesamteinsatz beträgt bis jetzt 7 DM. Wenn ROT kommt, erhält er 8 DM und hat wieder 1 DM gewonnen. Wenn SCHWARZ kommt, setzt er 16 DM.

Seine Strategie lautet also folgendermaßen: Er setzt zunächst 1 DM auf ROT (R). Solange er verliert, d.h., solange SCHWARZ (S) kommt, verdoppelt er jedesmal seinen Einsatz. Hier sind einige mögliche Spielserien: R, SR, SSR, SSSR, SSSSR, usw. Nach Abschluß jeder Spielserie hat er genau 1 DM gewonnen. Er gewinnt den Eindruck, daß er eigentlich nie verlieren kann und freut sich über seine neue Einnahmequelle.

Aufgabe 6.12: Schreibe eine Prozedur zum Ausdruck einer Tabelle der folgenden Art:

```
LETZTER   !  GESAMT-   !  AUS-      !  SERIEN-
EINSATZ   !  EINSATZ   !  ZAHLUNG   !  GEWINN
-----------------------------------------------
    1     !     1      !     2      !     1
    2     !     3      !     4      !     1
    4     !     7      !     8      !     1
    8     !    15      !    16      !     1
   16     !    31      !    32      !     1
 ...
```

Tabelle 6.2

Aufgabe 6.13: Wo liegt der Pferdefuß bei Gustavs Überlegung?

Die folgende Funktion ROULETTE gibt R (für ROT) oder S (für SCHWARZ) jeweils mit der Wahrscheinlichkeit 1/2 zurück

```
PR ROULETTE
 WENN  ( ZZ  2 ) = 0   DANN  RUECKGABE  "S
 RUECKGABE  "R
ENDE
```

Ein Beispiel im Direktbetrieb:

```
WH 20 [ DR ROULETTE ]
SRRSSSRSRSSSRRSRRRSR
```

Die folgende Prozedur simuliert jeweils eine Spielserie, druckt dabei die jeweiligen Einsätze aus und gibt als Funktionswert den Gesamteinsatz zurück.

```
PR ROULETTE.SERIE
 LOKAL  "GESAMTEINSATZ
 SETZE  "GESAMTEINSATZ  0
 ROULETTE.SERIE.HP  1
 RUECKGABE  :GESAMTEINSATZ
ENDE
```

```
PR ROULETTE.SERIE.HP  :EINSATZ
 SETZE  "GESAMTEINSATZ   :GESAMTEINSATZ + :EINSATZ
 DRUCKEZEILE  :EINSATZ
 WENN ROULETTE = "R  DANN RUECKKEHR
 ROULETTE.SERIE.HP  ( 2 * :EINSATZ )
ENDE
```

```
ROULETTE.SERIE
1
2
4
8
16
32
ERGEBNIS: 63
```

Wenn eine Serie sehr lang wird, wird es für Gustav gefährlich. Denn dann droht ihm das Geld auszugehen. Außerdem gibt es einen Höchstbetrag für den Einsatz, der nicht überschritten werden darf. In der folgenden Prozedur wird eine ganze Versuchsreihe von Roulette-Serien simuliert und jeweils der höchste (=letzte) Einsatz jeder Serie ausgedruckt. Der DRUCKEZEILE-Befehl in Zeile 3 der Prozedur ROULETTE.SERIE.HP wurde vorher gelöscht (z.B. mit der "Semikolon"-Methode; siehe Kapitel 3, Hinweis zu Aufgabe 3.6).

```
ROULETTE.SERIE.VERSUCHSREIHE  :ANZAHL
 WIEDERHOLE  :ANZAHL  [ DRUCKEZEILE  ROULETTE.SERIE ]
ENDE
```

```
ROULETTE.SERIE.VERSUCHSREIHE 10
3
1
63
3
15
3
1
31
1
7
```

Aufgabe 6.14: Schreibe eine Prozedur
RUSSISCHES.ROULETTE :OBERGRENZE ,
die fortlaufend immer wieder Roulette-Serien ausführt, bis der Einsatz die vorgegebene Obergrenze einmal überschreitet. Drucke jeweils aus, um die wievielte Roulette-Serie es sich handelt.

6.7 Wer kommt zum Höfleswetz-Turnier?

Zum Ende des Ferienprogramms der Stadt Reutlingen wird jeweils ein "Höfleswetz"-Fußballturnier ausgetragen, zu dem man sich zu Beginn der Ferien anmelden muß. Einige Schüler der Klasse 8c der Hap-Grieshaber-Schule beschließen, unter dem Namen "1.FC Schwabenbomber" eine Mannschaft zu bilden. Dazu wird eine (Kandidaten-) Liste von Spielern aufgestellt. Von den letzten Jahren her wissen die Schüler, daß manch einer der Kandidaten dann doch nicht erscheint; sei es, weil er

mit seinen Eltern in Urlaub ist, sei es, weil er sich den Fuß verstaucht hat, oder sei es, weil er es einfach vergessen hat. Deshalb versuchen die Schüler, möglichst viele Kandidaten auf ihre Liste zu bekommen. Schließlich haben sie eine Liste mit 15 Namen beisammen: Alex, Bernd, Chris, Dirk, Frank, Günter, Heinz, Jan, Matze, Oli, Peter, Ralf, Thomas, Uwe und Waggi.

Man weiß schon, wer ziemlich sicher kommt und wer eher ein unsicherer Kandidat ist. Die Chancen zu kommen werden folgendermaßen eingeschätzt:

```
Alex:     95 %
Bernd:    90 %
Chris:    70 %
Dirk:     65 %
Frank:    80 %
Günter:   55 %
Heinz:    60 %
Jan:      70 %
Matze:    90 %
Oli:      90 %
Peter:    45 %
Ralf:     55 %
Thomas:   70 %
Uwe:      80 %
Waggi:    85 %
```

Tabelle 6.3: Teilnahmewahrscheinlichkeiten

Die Schüler sind natürlich daran interessiert zu wissen, wie viele der Kandidaten wohl schließlich am Turnier teilnehmen werden. Genau kann man das natürlich nicht voraussagen. Uwe hat an einer Computer-AG teilgenommen und dabei einige Simulationsprogramme kennengelernt. Er kommt deshalb auf die Idee, für das Fußballturnier ein kleines Simulationsprogramm zu schreiben.

Er ist sich darüber im klaren, daß er zuerst die Tabelle der Teilnahmewahrscheinlichkeiten in geeigneter Weise nach "Logo" übertragen muß. Uwe löst das Problem, indem er eine Funktion KANDIDATEN schreibt, die genau die Tabelle 6.3 (in Listenform) ausgibt:

```
PR KANDIDATEN
 RUECKGABE  [ [ ALEX  0.95 ]    [ BERND  0.9 ]  _____
 ___________  [ CHRIS  0.7 ]    [ DIRK  0.65 ]  _____
 ___________  [ FRANK  0.8 ]    [ GUENTER 0.55 ] ____
 ___________  [ HEINZ 0.6 ]     [ JAN  0.7 ]  _______
 ___________  [ MATZE  0.9 ]    [ OLI  0.9 ]  _______
 ___________  [ PETER  0.45 ]   [ RALF  0.55 ] ______
 ___________  [ THOMAS  0.7 ]   [ UWE  0.8 ]  _______
 ___________  [ WAGGI  0.85 ]   ]
ENDE
```

Als nächstes schreibt er eine Funktion TEILNEHMER :KL, mit der er durch eine Simulation aus der (Kandidaten-) Liste :KL die Teilnehmer ermittelt.

```
PR TEILNEHMER  :KL
 WENN LEER?  :KL  RG  [ ]
 PRUEFE  ( ZZ 1000 ) / 1000  <  LZ ER :KL
 WW  RG  ME ( ER ER :KL )  TEILNEHMER  OE  :KL
 WF  RG  TEILNEHMER  OE  :KL
ENDE
```

Wie du siehst, macht es Uwe keinen Spaß, lange Schlüsselwörter einzutippen. Die Langformen der hier benutzten Kurzformen sind: ERSTES (ER), LETZTES (LZ), MITERSTEM (ME), OHNEERSTES (OE) und RUECKGABE (RG). Das erste Element der Kandidatenliste ist eine Paarliste; das erste vom ersten ist der Name des Kandidaten und das letzte vom ersten ist seine Teilnahmewahrscheinlichkeit. Ein Test ergibt:

TEILNEHMER KANDIDATEN
ERGEBNIS: [ALEX BERND DIRK FRANK MATZE OLI RALF UWE WAGGI]

Aufgabe 6.15: Dirk wundert sich, daß dieser Aufruf nicht TEILNEHMER :KANDIDATEN lauten mußte. Erkläre ihm, warum.

Wenn man nur wissen will, wie viele Teilnehmer von der Simulation herausgepickt werden, so braucht man sich nur die Länge (=Elementezahl) der Ergebnisliste ausgeben zu lassen:

LAENGE TEILNEHMER KANDIDATEN
ERGEBNIS: 12

Aufgabe 6.16: Probiere folgendes im Direktbetrieb aus:
WH 100 [DZ TEILNEHMER KANDIDATEN]
WH 100 [DZ LAENGE TEILNEHMER KANDIDATEN]

Nach einiger Zeit kommen Uwe Bedenken, ob die Simulation wirklich realistisch ist. Denn Bernd und Oli sind doch Zwillinge: mit ziemlicher Sicherheit kommen sie entweder beide, oder es kommt keiner von ihnen. Außerdem ist Peter sehr vergeßlich. Die anderen haben Alex deshalb gebeten, Peter kurz vor dem Turnier nochmals daran zu erinnern. Alex ist sehr zuverlässig. Man kann sich praktisch hundertprozentig darauf verlassen, daß er das auch tut.

Aufgabe 6.17: Baue Uwe's Simulationsprogramm so aus, daß diesen Umständen Rechnung getragen wird.

Aufgabe 6.18: Die Textilfirma Wuschelweich bekommt einen Großauftrag über Pullover, der innerhalb einer Woche ausgeliefert werden muß. Es ist enorm wichtig, daß der Liefertermin pünktlich eingehalten wird. Die Firma hat 20 gleichartige Strickmaschinen, von denen jede eine Ausfallwahrscheinlichkeit von 5 % pro Tag hat. Die Maschinen arbeiten unabhängig voneinander. Der Liefertermin kann nur dann eingehalten werden, wenn in dieser Woche nie mehr als 3 Maschinen pro Tag ausfallen. Der Chef möchte vorher wissen, wie groß die Chancen dafür sind und ob er eventuell noch einige Maschinen anmieten sollte. Er gibt dir den Auftrag, ein Programm zu schreiben, mit dem man die *Maschinenausfälle* simulieren kann. Das Ergebnis soll folgendermaßen aussehen (0 bedeutet "kein Ausfall", 1 bedeutet "Ausfall"):

Tag	M1	M2	M3	...	M19	M20	Gesamtausfälle pro Tag
1	0	0	1		0	0	1
2	0	1	0		1	0	2
3	0	1	0		0	0	1
4	0	0	0		0	0	0
5	0	0	0		0	1	1
...							

Tabelle 6.4: Maschinenausfälle

Aufgabe 6.19: Das Thema "Simulationen" ist unerschöpflich. Hier sind einige weitere Anregungen für Simulationsprogramme:
(a) Das *Sammlerproblem*: In Haferflocken-Packungen sind manchmal "Sammelmarken" (z.B. Pennies mit den Ziffern 1, 2,...,6) enthalten. Wenn man einen *vollständigen Satz* hat, der aus allen 6 möglichen Pennies besteht, kann man sich beim Kaufmann ein kleines Geschenk abholen.
(b) *Irrfahrten*: Ein Teilchen kann sich pro Sekunde in eine der vier Richtungen "nach oben", "nach rechts", "nach unten" und "nach links" bewegen. Simuliere die "Irrfahrt eines solchen Teilchens mit dem Igel.
(c) *Rosinenbrötchen*: In einen Teig für 100 Rosinenbrötchen werden 500 Rosinen gegeben. Das Ganze wird gut gemischt und dann gebacken. In wie vielen Brötchen ist mit 0, 1, 2, 3, 4, ... , 10, ... Rosinen zu rechnen?
(d) *Tischtennis*: Wie viele Angaben gibt es durchschnittlich noch, wenn es bei einem Tischtennis-Spiel 20 : 20 steht?
(e) *Mehrfach-Geburtstage*: Wie groß sind die Chancen, daß in einer Klasse von 20 (bzw. 21, 22, ..., 35) Schülern (ohne Zwillinge) mindestens zwei am selben Tag Geburtstag haben?
(f) *Lotto- und Toto-Ziehungen.*

6.8 Stabdiagramme (Histogramme)

Wir wollen uns jetzt der Aufgabe zuwenden, die in Abschnitt 6.4 erzeugten Häufigkeitstabellen in Stabdiagramme (Histogramme) umzusetzen. Dabei knüpfen wir an das an, was wir in Kapitel 5, Abschnitt 5.2 über Funktionsschaubilder erarbeitet haben.

Die Routinen zur Festlegung des Bildausschnitts, zum Zeichnen des Rahmens und der Achsen können genau so übernommen werden, wie sie in Kapitel 5 erstellt wurden. Wir hatten diese Routinen unter dem Namen ACHSENDATEI gesondert abgespeichert. Nach Eingabe des Befehls LADE "ACHSENDATEI stehen sie uns wieder zur Verfügung.

In den folgenden Prozeduren ist :TAB eine Paarliste der Art, wie sie von der Funktion HAEUFIGKEITSTABELLE (siehe Abschnitt 6.4) zurückgegeben wird. Natürlich können auch Paarlisten eingegeben werden, die in ganz anderen Zusammenhängen entstehen. Die Prozedur HISTOGRAMM hat im wesentlichen dieselben Parameter wie die Prozedur FUNKTIONS.SCHAUBILD. An Stelle der Funktion haben wir jetzt natürlich die aus den Zahlenpaaren bestehende Tabelle. Der Schrittweiten-Parameter entfällt, da wir mit der festen Schrittweite 1 arbeiten.

HISTOGRAMM soll wie in der folgenden "Demo"-Prozedur aufgerufen werden. Man kann die Tabelle auch direkt eingeben, ohne vorher die Hilfsvariable

TEST.TABELLE erzeugt zu haben. Auch der Aufruf im Direktbetrieb ist selbstverständlich möglich.

```
PR DEMO.HISTOGRAMM
 LOKAL  "TEST.TABELLE
 SETZE  "TEST.TABELLE  [  [ 1  0.5  ]  [ 2  0.6 ]  ____
 ______          [ 3  0.4  ]  [ 4  1.2  ]  [ 5  0.7 ]  ____
 ______          [ 6  0.75 ]  [ 7  0.45 ]  ]
 HISTOGRAMM  :TEST.TABELLE  ( - 1 )  8  ( - 0.1 )  1.5
ENDE
```

Die Demo-Prozedur erzeugt das folgende Schaubild.

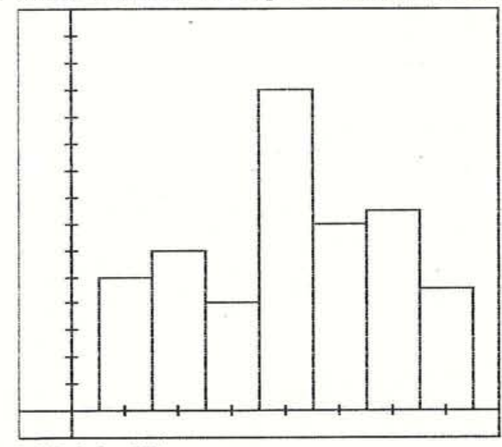


Bild 6.3: Histogramm

Nun zu den Prozeduren der HISTOGRAMM-Gruppe. Mit unseren Vorkenntnissen aus Kapitel 5 erklären sie sich fast von selbst. Die trivialsten Dinge machen, wie so oft, auch hier die größten Schwierigkeiten; so zum Beispiel die Tatsache, daß eine Histogramm-Säule möglicherweise nicht in den vorgegebenen Bildausschnitt paßt. Hierfür gibt es zwei Möglichkeiten. Zum ersten kann es passieren, daß einer der "Fußpunkte" der Säule (damit sind die Schnittpunkte der Säule mit der x-Achse gemeint) nicht im Bildausschnitt liegt. In diesem Fall wird keine Säule gezeichnet. Zum zweiten ist es möglich, daß zwar die Fußpunkte, nicht aber das "Obergeschoß" bzw. der "Keller" der Säule in den Bildausschnitt passen. Wir lösen dieses zweite Problem so, daß zur Kennzeichnung solcher Stäbe von oben nach unten ein großes X durch den Stab gezeichnet wird. Dies läßt sich relativ leicht bewerkstelligen. Es kann durchaus "schönere" Möglichkeiten zur Kennzeichnung dieses Sachverhalts geben. Wenn dir eine andere Lösung besser gefällt, brauchst du nur die Prozedur ERSATZ.SAEULE entsprechend abzuändern. Das folgende Schaubild entsteht durch den Aufruf HISTOGRAMM :TEST.TABELLE (-1) 6 (-0.1) 0.9 , wobei :TEST.TABELLE eine Liste mit den in DEMO.HISTOGRAMM gegebenen Werten ist.

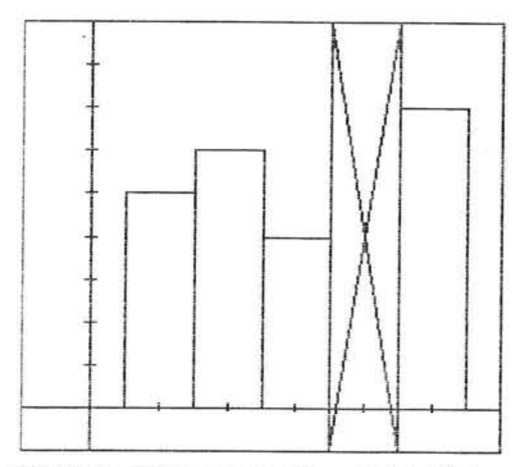


Bild 6.4: Histogramm mit zu hoher Säule

```
PR HISTOGRAMM  :TAB  :XMIN  :XMAX  :YMIN  :YMAX
 ACHSEN  :XMIN  :XMAX  :YMIN  :YMAX
 HISTOGRAMM.HP  :TAB
ENDE

PR HISTOGRAMM.HP  :TAB
 WENN  LEER? :TAB  DANN  RUECKKEHR
 SAEULE  ( ERSTES ERSTES :TAB ) ( LETZTES ERSTES :TAB )
 HISTOGRAMM.HP  OHNEERSTES  :TAB
ENDE

PR SAEULE  :X  :Y
 WENN X.AUSSERHALB? TRANSX ( :X - 0.5 )  DANN RUECKKEHR
 WENN X.AUSSERHALB? TRANSX ( :X + 0.5 )  DANN RUECKKEHR
 WENN Y.AUSSERHALB? TRANSY :Y  DANN
           ERSATZ.SAEULE :X :Y   RUECKKEHR
 STIFTHOCH
 AUFXY  TRANSX ( :X - 0.5 )  TRANSY 0
 STIFTAB
 AUFY  TRANSY  :Y
 AUFX  TRANSX  ( :X + 0.5 )
 AUFY  TRANSY 0
ENDE

PR ERSATZ.SAEULE  :X  :Y
 STIFTHOCH
 AUFXY  TRANSX (  :X - 0.5 )  :YBMIN
 STIFTAB
 AUFY  :YBMAX
 AUFXY  TRANSX  ( :X + 0.5 )  :YBMIN
 AUFY  :YBMAX
 AUFXY  TRANSX  ( :X - 0.5 )  :YBMIN
ENDE
```

```
PR X.AUSSERHALB?  :XB
 WENN :XB < :XBMIN  DANN RUECKGABE "WAHR
 WENN :XB > :XBMAX  DANN RUECKGABE "WAHR
 RUECKGABE "FALSCH
ENDE

PR Y.AUSSERHALB?  :YB
 WENN :YB < :YBMIN  DANN RUECKGABE "WAHR
 WENN :YB > :YBMAX  DANN RUECKGABE "WAHR
 RUECKGABE "FALSCH
ENDE
```

Es können natürlich insbesondere auch die von der Prozedur HAEUFIGKEITS-TABELLE erzeugten Tabellen in die Histogramm-Prozedur eingespeist werden:

```
HISTOGRAMM HAEUFIGKEITSTABELLE ___
___ ( NAEGEL.VERSUCHSREIHE 100 0.2 200 ) ( -1 ) 150 ( -1 ) 25
```

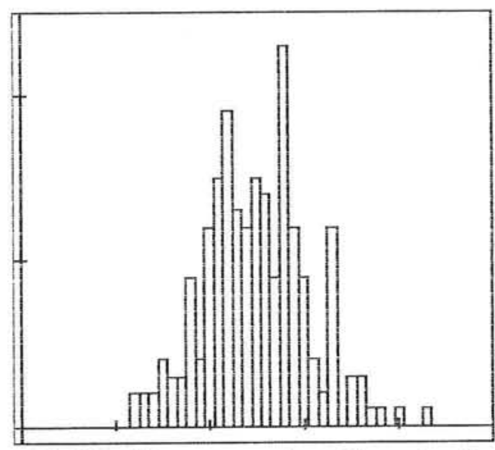


Bild 6.5: Histogramm einer Versuchsreihe

Aufgabe 6.20: Schreibe Funktionen
X.MINIMUM :PAARLISTE,
X.MAXIMUM :PAARLISTE,
Y.MINIMUM :PAARLISTE,
Y.MAXIMUM :PAARLISTE,
mit denen du den jeweils kleinsten und den größten x- und y-Wert in der Paarliste ermitteln kannst. Verwende diese Funktionen so, daß das zu zeichnende Histogramm mit Sicherheit in den Bildausschnitt paßt; zum Beispiel folgendermaßen:

```
PR DEMO.2
 LOKAL  "HT
 SETZE  "HT  HAEUFIGKEITSTABELLE ___
             NAEGEL.VERSUCHSREIHE  100  0.2  200
 HISTOGRAMM  :HT  ( -1 )  ( ( X.MAXIMUM  :HT ) + 1 ) ___
 ___    ( -1 )  ( ( Y.MAXIMUM  :HT )  +  1 )
ENDE
```

Aufgabe 6.21: Baue die Histogramm-Prozedurgruppe nach deinen Wünschen weiter aus (zum Beispiel: Skalierung der Achsen, Fehlermeldungen, andere Lösungen von ERSATZ.SAEULE,...).

Aufgabe 6.22: Schreibe eine Prozedur, mit der du mehrere Histogramme gleichzeitig abbilden kannst; etwa so wie in der folgenden Abbildung. Dort sind die *Summenwahrscheinlichkeiten* beim Würfeln mit einem, zwei, drei und vier Würfeln dargestellt.

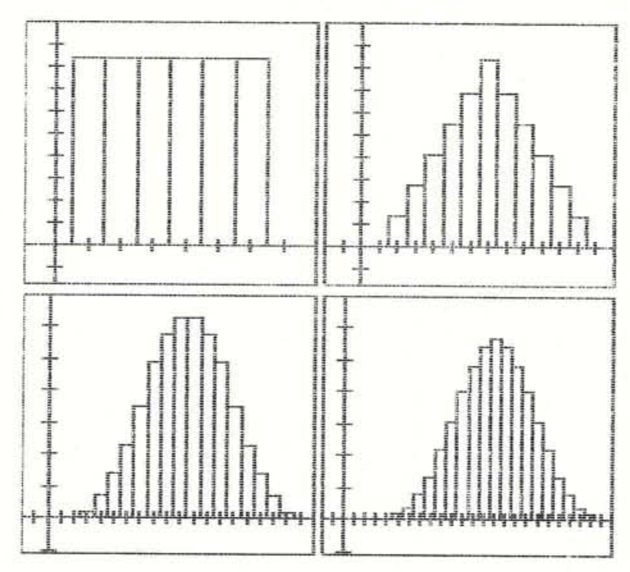


Bild 6.6: Summenwahrscheinlichkeiten beim Würfeln

Stichwortverzeichnis